Introduction to Cloud Blueprints

Describing a Set of Related Cloud Resources

Jan. 30, 2013

Contents

Introduction 1
  Uses of Blueprints 2
Concepts 3
  Enterprise Manager Cloud Concepts 3
    Oracle Cloud API 3
    Oracle Cloud Resource Model 3
    Cloud Resource Types and Attributes 3
    Resource Containment 3
  Blueprint Concepts 4
    Input Parameters 4
    Resource 4
    Outputs 4
    Intrinsic Functions 4
    Named Literals 4
    Dependencies 4
Deploying a Blueprint 5
  Usage 5
  Blueprint Deployment Processing 5
Blueprint Examples 9
  Blueprint Structure and Basics 9
    Simple Blueprint 9
    Simple Resource: Database Service Instance 10
    Intrinsic Functions 11
    Simple Resource with Parameter 11
    Data Section (Named Literals) 12
  Putting It All Together – Multiple Interdependent Resources 13
    Macro Section 15
    Attributes of Created Resources (Dependencies) 15
  Visual Depiction of Blueprint Processing 16
Closing 20
References 20

Introduction

This document introduces ‘cloud blueprints’ (more briefly, ‘blueprints’) for the Oracle Private Cloud, both how to use them and how to create your own.

Blueprints are used to describe a desired set of inter-related cloud resources. Like architectural blueprints, they describe *what* you want including how they are configured to interact with each other, but not *how* to build them. For instance, a blueprint doesn't describe the order in which to create the components. Rather, the blueprint orchestration logic figures that out based on inter-resource dependencies.

As an example, suppose you want to create a set of interacting cloud resources such as a WebLogic server instance, an application, and a database. To do so, you must first create the database and WebLogic server instance, deploy the application, and create a JEE datasource that is to be used by the WebLogic server to connect to the database.

You could perform all these operations manually, through the Enterprise Manager Cloud Self Service Portal. You would request creation of the WLS server and database and wait for either to complete. Periodically, you would check the status of the creation requests. Once the WLS server is created, you could deploy the application. When both the WLS server and database are created, you could create the JEE datasource.

Alternatively, you can use a blueprint that describes the four cloud resources to automate the process. You request instantiation of the blueprint and provide any input parameter values required by the blueprint. The blueprint initiates the creation of the resources and monitors the creation process to ensure that the dependent resources are automatically created as soon as the required resources are created.

The rest of this document introduces the blueprint concepts including how to deploy an existing blueprint as well as how to write your own.

Also available is a user manual [BP_REF] that goes into much greater detail on blueprints.

Uses of Blueprints

A blueprint can be used to automate the creation of service instances. An EM_SSA_USER can use blueprints for various reasons:

- To create an application composed of several service instances and related cloud resources.
- To create such sets several times (possibly with small variations).
- To facilitate instance creation for other EM_SSA_USERS.
- To eliminate the manual interactive steps that would otherwise be needed to create the set of instances
- You want a textual representation, e.g. to place it under version control or so that you can give it to someone else in a form he can review and modify.

To summarize, blueprints allow you to automate the creation of a set of service instances.

For example, the Quality Assurance team in an enterprise needs to allocate and release resources required to test a Web application. Instead of manually creating the service instances using the Enterprise Manager Cloud Self Service application, a blueprint can be used to perform this task. One person authors a blueprint so that all QA engineers can simply invoke the blueprint and enter a few input parameter values, after which the resources are created. Each user can watch as the blueprint processor displays the status for creation of each resource.

Another example illustrates a blueprint's use to address simplicity and consistency concerns. An IT shop has a service template that accepts 8 input parameters. For a specific group of users, the same set of values should be used for 6 of those 8 parameters. A simple blueprint accepts 2 parameters and uses the template to instantiate the instances with the other 6 parameters consistently defined.

Concepts

Enterprise Manager Cloud Concepts

The concepts described in this section are Enterprise Manager cloud concepts. They are not introduced as *part of* blueprints but are concepts *used by* blueprints. Since they form the basis for blueprints, they are summarized below.

Oracle Cloud API

The Oracle Cloud API [CRMA] defines a RESTful programming interface to consumers of IaaS, MWaaS, and DBaaS based on Oracle's solution stack. It's the Oracle Cloud API that the blueprint processor uses to create cloud resources based on what's specified by the blueprint.

Oracle Cloud Resource Model

The Oracle Cloud Resource Model [CRMA] details the types of resources one can manipulate via the Oracle Cloud API and, for each type, both its attributes and the operations that can be performed on instances of a resource type.

Cloud Resource Types and Attributes

The cloud resource model specifies a set of attributes that are common to all cloud resources, such as *uri* (its URI) and *resource_status* (with values such as READY and CREATING). The model also defines a set of cloud resource types and their attributes. Some resource types discussed later in this document are:

- **DbPlatformInstance**: A database platform instance is created using a template. It has attributes such as *zone* (location for DBaaS instance) and *params,*(to specify username and password).
- **JavaPlatformInstance**: A JEE server instance is also created using a template. Similar to DbPlatformInstance, a JavaPlatformInstance has attributes like *zone.* An example of a MWaaS-specific attribute is *application_instance_deployments ,* that identifies all applications deployed to the instance.
- **ApplicationInstanceDeployment**: A resource type that represents an application deployment to a JavaPlatformInstance.
- **Datasource**: A resource type that represents a datasource of a JavaPlatformInstance. It is contained in a JavaPlatformInstance and refers to a DatabaseInstance.

Resource Containment

A cloud resource can *contain* other resources. For instance, a Datasource of a JavaPlatformInstance is *contained* in the JavaPlatformInstance, and a template *contains* all service instances that were created using that template.

Every resource you create will be contained in a parent cloud resource. As part each resource definition in your blueprints, you will identify the resource's container.

Blueprint Concepts

A blueprint generally describes one or more cloud resources to be created.

A user *deploys* a blueprint to create the resources described by the blueprint, at which time he provides any input parameter values used by the blueprint.

To create each resource, the blueprint specifies its attribute values, which may be hard-coded, come from blueprint input parameters, or gleaned from other resources. In instantiating all the resources, the blueprint system determines the resource dependencies and uses this to order the resource creation and/or configuration required to properly create the instances described by a blueprint.

Input Parameters

A blueprint can define input 'parameters'. Each parameter definition specifies a name, datatype and optional specifications such as default value. A user who deploys a blueprint must specify the value for all parameters not having a default.

Resource

A blueprint 'resource' defines how to construct a cloud resource. It specifies a set of attributes and the parent cloud resource that is to contain the newly created resource. Each resource in a blueprint also has a name, which must be unique within a blueprint.

Outputs

If *outputs* are specified, the specified values are displayed when blueprint deployment completes. Output parameters can be used to display information derived during deployment such as to inform the end user of the URL of a JEE application deployment.

Intrinsic Functions

The blueprint language includes a set of predefined functions, illustrated below.

Named Literals

As a programming convenience, blueprints can include a Data section. This is commonly used to specify symbolic names for literal values.

Dependencies

Blueprint deployment is done by creating resources in parallel when possible, but a resource that depends on another resource can't be created until the latter resource is created.   Such dependencies are often implicit, but blueprint processing identifies dependencies and orchestrates the overall steps.

Deploying a Blueprint

Usage

Prerequisites:

- You have the blueprint file
- You have installed the blueprint processor software

A blueprint file is a text file in which the author has used the blueprint language to describe what is required. You deploy the blueprint by running the blueprint processor. For instance, on Windows you would use a command as follows:

```
bp_processor.py myfile.yml -u myname  -c https://myhost/em/cloud
```

In this example, the -u option specifies the Enterprise Manager user id. If the password is not specified here, you are prompted for it when you execute the blueprint file. The -c option is used to specify the cloud URL. There are numerous other command line options. To view a description of supported options, enter

```
bp_processor.py myfile.yml -h
```

Blueprint Deployment Processing

When a blueprint is deployed, the runtime logic processes the input parameters and orchestrates the creation of resources, doing so in parallel when possible. It also monitors progress and keeps the user informed.

The output you see will depend on the blueprint and your environment. The example below shows the use of a blueprint that deploys a Weblogic server, application, database, and JEE Datasource. The output from running this blueprint should give you a sense for the blueprint processing steps.

```
C:\Users\myname\Dropbox\Code\blueprints>bp_processor.py xyzApp.yml -c
https://... -g deployment_report
...
Cloud user id: ssa_user1
Password:
```

The command you entered above specifies the cloud URL (via the -c option) and a directory into which to place the optional deployment report (the -g option). You then entered the credentials as prompted to proceed.

```
Blueprint Processor - Invocation Summary
----------------------------------------
   Cloud URI:                 https://...
   User:                      ssa_user1
   Blueprint file:            xyzApp.yml
   Timeout:                   90 minutes, 0 seconds
   Refresh frequency:         15 seconds
   Inputs:
   Pause points:             (none)
   Debug logging:            False
   Instance name:            default_instance_name
   Graphical report dir:     deployment_report
   Versions:
      Blueprint processor:   12.1.0.5, 10-Oct-2012
      Cloud protocol:        10001

14:24:59  INFO: Connecting to cloud: https://....
```

(Most of the information in the invocation summary reflects default values used because you didn't specify the corresponding options.)

Because the blueprint defines input parameters, you are then prompted to provide values:

```
Input Parameter Value Entry
---------------------------
   Zone to use for db (Zone1):
   Password to use for db (welcome1):
```

In this example, you pressed **Enter** to accept the default value for zone, Zone1 and then entered a password.

Once all the necessary information has been provided, the blueprint processor starts creating the resources and monitoring the progress. This is depicted in a "vertical timeline" so you can observe the progress. Each resource to be created is represented by a column. As the state transitions for each resource occur, they are noted in the corresponding column of the vertical timeline.

First, the list of all possible states, with their abbreviations, is printed. The states reflect the processing phases and the outcome of each phase.

```
14:25:03  INFO:
14:25:03  INFO: Resource State Timeline
14:25:03  INFO: -----------------------
14:25:03  INFO:   State Key:
14:25:03  INFO:      e : Evaluating
14:25:03  INFO:      ep: Evaluation pending. (See right side for pendee)
14:25:03  INFO:      es: Evaluation succeeded, creation requested
14:25:03  INFO:      EF: Evaluation failed
14:25:03  INFO:      c : Creating
14:25:03  INFO:      CF: Creation failed
14:25:03  INFO:      CS: Creation succeeded. State = READY
14:25:03  INFO:
```

To process any resource definition, the first step is to evaluate the expressions of the definition that describe the resource (State Key: e). In some cases, evaluation must be delayed (State Key: ep). Once fully evaluated, a creation request is made (State Key: c) and the processing for that resource is successfully completed (State Key: CS).After the State Key is printed, the resource creation timeline appears:

To process any resource definition, the first step is to evaluate the expressions of the definition that describe the resource (Cloud state: e). In some cases, evaluation must be delayed (state key: ep). Once fully evaluated, a creation request is made (state key: c). This make take some time, but if all goes well the processing for that resource completes successfully (state key: CS).

After the State Key is printed, the resource creation timeline starts to appear:

```
14:25:03  INFO:              MyApp
14:25:03  INFO:             /
14:25:03  INFO:            /    MyDB
14:25:03  INFO:           /    /
14:25:03  INFO:          /    /    MyDS
14:25:03  INFO:         /    /    /
14:25:03  INFO:        /    /    /    MyWebServer
14:25:03  INFO:       /    /    /    /
14:25:03  INFO:   ---------------------
14:25:03  INFO:   |    |    |    | e  |
14:25:04  INFO:   |    |    |    | es |
14:25:10  INFO:   |    |    |    | c  |
14:25:10  INFO:   |    | e  |    | .  |
14:25:12  INFO:   |    | es |    | .  |
14:25:17  INFO:   |    | c  |    | .  |
14:25:17  INFO:   |    | .  | e  | .  |
14:25:17  INFO:   |    | .  | ep | .  | Awaiting creation of MyWebServer
14:25:17  INFO:   | e  | .  |    | .  |
14:25:17  INFO:   | ep | .  |    | .  | Awaiting creation of MyWebServer
14:25:33  INFO:   |    | .  |    | .  |
14:25:50  INFO:   |    | .  |    | .  |
.....
14:41:18  INFO:   |    | .  |    | .  |
14:41:33  INFO:   |    | .  |    | CS |
14:41:33  INFO:   |    | .  |    |====|
14:41:44  INFO:   |    | .  | ep |    | Awaiting creation of MyDB
14:41:57  INFO:   | es | .  |    |    |
14:42:00  INFO:   | c  | .  |    |    |
14:42:17  INFO:   | .  | .  |    |    |
....
14:44:44  INFO:   | .  | .  |    |    |
14:45:16  INFO:   | CS | .  |    |    |
14:45:16  INFO:   |====| .  |    |    |
14:45:32  INFO:   |    | .  |    |    |
14:45:47  INFO:   |    | .  |    |    |
14:54:17  INFO:   |    | .  |    |    |
14:54:17  INFO:   |    | CS |    |    |
14:54:17  INFO:   |    |====|    |    |
14:54:18  INFO:   |    |    | es |    |
14:54:19  INFO:   |    |    | c  |    |
14:54:33  INFO:   |    |    | CS |    |
14:54:33  INFO:   |    |    |====|    |
14:54:33  INFO:   ---------------------
14:54:33  INFO:
```

You can see that the timeline is vertical and the four resources are MyApp, MyDB, MyDS, and MyWebServer.

The MyWebServer resource definition is evaluated (state key: e) first. When this is successful, the creation process begins (state key: c). Parallely, the MyDB resource is evaluated and the creation process is initiated. Then the MyDS resource is evaluated and the blueprint processor determines that the evaluation cannot be completed until the MyWebServer resource is created. The same process is applicable to the MyApp resource. When the MyDB and MyWebServer resources are successfully created (state key: CS), the creation process for MyApp and MyDS resources can proceed.

Next, the Outputs section of our example blueprint is processed:

```
14:54:33  INFO: Output Processing
14:54:33  INFO: -----------------
14:54:33  INFO:
14:54:33  INFO: Output values specified: 1
14:54:49  INFO:    Value of URL: {u'ms_1': u'http://...}
14:54:49  INFO:
```

In the example above, you can see that the blueprint specifies one output value named “URL” and a value is represented with https://….

When all the resources have been successfully created, the blueprint processor summarizes the results. This includes the processing summary for each requested resource as well as the timing information for each resource and the overall run:

```
14:54:49  INFO: Blueprint Processing Summary
14:54:49  INFO: ----------------------------
14:54:49  INFO:
14:54:49  INFO: Resource State Summary:
14:54:49  INFO:    MyWebServer: READY
14:54:49  INFO:       URI: /em/cloud/jaas/javaplatforminstancerequest/163
14:54:49  INFO:       Cloud resource state: READY
14:54:49  INFO:       Timing info:
14:54:49  INFO:         Creation start:     14:25:04
14:54:49  INFO:         Creation end:       14:41:33
14:54:49  INFO:         Duration:           16 minutes, 29.6 seconds
14:54:49  INFO:    MyDB: READY
14:54:49  INFO:       URI: /em/cloud/dbaas/dbplatforminstance/byrequest/164
14:54:49  INFO:       Cloud resource state: READY
14:54:49  INFO:       Timing info:
14:54:49  INFO:         Creation start:     14:25:12
14:54:49  INFO:         Creation end:       14:54:17
14:54:49  INFO:         Duration:           29 minutes, 5.7 seconds
14:54:49  INFO:    MyDS: READY
14:54:49  INFO:       URI: /em/cloud/jaas/datasourcerequest/QA_app_DS@201
14:54:49  INFO:       Cloud resource state: READY
14:54:49  INFO:       Timing info:
14:54:49  INFO:         Creation start:     14:54:18
14:54:49  INFO:         Creation end:       14:54:33
14:54:49  INFO:         Duration:           0 minutes, 15.3 seconds
14:54:49  INFO:    MyApp: READY
14:54:49  INFO:       URI:
/em/cloud/jaas/applicationinstancedeploymentrequest/myApp@181
14:54:49  INFO:       Cloud resource state: READY
14:54:49  INFO:       Timing info:
14:54:49  INFO:         Creation start:     14:41:57
14:54:49  INFO:         Creation end:       14:45:16
```

```
14:54:49  INFO:         Duration:           3 minutes, 18.9 seconds
14:54:49  INFO:
14:54:49  INFO: Timing Summary (seconds):
14:54:49  INFO:    Client-side CPU time: 0 minutes, 6.474 seconds
14:54:49  INFO:    Elapsed time:
14:54:49  INFO:       Processing time:    29 minutes, 58.0 seconds
14:54:49  INFO:       Paused time:        0 minutes, 1.9 seconds
14:54:49  INFO:       Total elapsed time: 29 minutes, 59.9 seconds
14:54:49  INFO:
14:54:49  INFO: Graphical Report Generation
14:54:49  INFO: -------------------------------
14:54:49  INFO:
14:55:37  INFO: Graphical report generated: deployment_report/bp_report.html

C:\Users\myname\Dropbox\Code\blueprints>
```

Blueprint Examples

This section illustrates the use of the blueprint concepts and syntax by guiding the reader through progressively more complex examples of blueprints. If you don't plan to author any blueprints, you may choose to skip this section.

Blueprint Structure and Basics

A cloud blueprint specifies a set of desired cloud resources and represents these resources via a text file. Blueprints leverage a standard for easily readable data-structured text called YAML[1].

Simple Blueprint

A blueprint is a document that you can think of as containing sections. The simplest useful blueprint specifies only the 'Resources' section and a single resource. In this example, the resource is a database defined by a template.

```
Resources:
  MyDB:
    Type: application/oracle.com.cloud.common.DbPlatformInstance+json
    Container: ...   (refers to db template)
    Properties: ... (provides properties of db)
```

The above blueprint defines one blueprint resource named MyDB. The Type entry specifies the media type for 'database' as defined by the Cloud Resource Model API. The Container entry identifies the parent cloud resource to contain the newly created object. (Per [CRMA], all cloud resources are created by adding them to existing containers.) Being a database service instance, it will be created vua the database template used to create the service. How to specify the container will be shown later, so we just use ellipses here.   Similarly, the data required by that template is specified in the Properties entry and shown later.

Now let's add an 'Inputs' section…

```
Inputs:
```

---

[1] YAML is a standard notation, like XML and JSON. As with JSON, YAML is used to represent information via lists, dictionaries, and nesting. These concepts are sufficient to capture all blueprint semantics.

```
  DbZone:
    Type: String
    DefaultValue: Zone1
    Prompt: Zone to use for db
  DbPassword:
    Type: String
    DefaultValue: welcome1
    Prompt: Password to use for db
    Sensitive: True
Resources:
  MyDB:
    Container: ...
    Properties: ...
```

When the user requests deployment of this blueprint, he provides a value for the DbZone parameter or takes the default specified by the blueprint (Zone1). After input processing, the parameter values can be referenced from other parts of the blueprint, in particular to provide attribute values needed to create resources. More on this later.

Simple Resource: Database Service Instance

In a blueprint, one uses *blueprint resources* to describe the *cloud resources* to create. For each blueprint resource, the information required by the Cloud Resource Model is provided.

The following blueprint specifies a single resource to create a database service instance:

```
Resources:
  MyDB:
    Type: application/oracle.com.cloud.common.DbPlatformInstance+json
    Container: ...
    Properties:
      zone: ...
      name: jbName
      params:
        username: app_user
        password: change_me
```

Each resource definition in a blueprint specifies a name, a Container into which to add the resource, and Properties used to specify the characteristics of what to create. In this case:

- The ***name*** of the blueprint resource definition is MyDB. The name is used in the scope of a blueprint, e.g. to inform the user deploying a blueprint about progress for each resource. In more complex cases, we'll see that the name can be referenced elsewhere within a blueprint.
- The **Container** entry specifies the URI of the container to which the new resource will be added. To create a database service instance, we identify the service template that corresponds to the kind of database we want. (We'll see how to do that when we introduce the topic of intrinsic functions.)
- The **Properties** entry specifies values needed to create the resource. In this case, the model requires that we specify *zone, name,* and *params* properties. These specify the zone in which the instance is to be created, its name, and a list of name/value pairs required by the selected template.

Intrinsic Functions

To operate on data, blueprints support the use of intrinsic functions. All function names begin with “f_” and are invoked with a list of arguments.

Continuing the above example, we use two intrinsic functions, to return the URI of the desired container and the desired zone.

```
Resources:
  MyDB:
    Type: application/oracle.com.cloud.common.DbPlatformInstance+json
    Container:
      f_getTemplateURI:
        - Small Database Service Template_automation_VIMAL_si
        - dbaas
    Properties:
      zone:
        f_getZoneURI: ...
      name: jbName
      params:
        username: app_user
        password: ...
```

As you can see, the f_getTemplateURI function takes 2 arguments, the name of the template and its service type. The current supported service types are: dbaas, jaas, and iaas (Database-, Java-, and Infrastructure-as-a-Service). The f_getZoneURI is analogous to f_getTemplateURI but for zones.

Other intrinsic functions will be introduced below. The full set of functions is described in [BP_REF].

Simple Resource with Parameter

To the above example blueprint, we now add the use of 2 parameters. By doing so, the user who deploys the blueprint, can specify which zone and password to use.

```
Inputs:
  DbZone:
    Type: String
    DefaultValue: Zone1
    Prompt: Zone to use for db
  DbPassword:
    Type: String
    DefaultValue: welcome1
    Prompt: Password to use for db
    Sensitive: True
Resources:
  MyDB:
    Type: application/oracle.com.cloud.common.DbPlatformInstance+json
    Container:
      f_getTemplateURI:
        - Simple DB Template
        - dbaas
    Properties:
      zone:
        f_getZoneURI:
          - f_path:
            - "Inputs.DbZone.Value"
          - dbaas
      params:
        username: app_user
        password:
          f_path:
            - 'Inputs.Password.Value'
```

The Inputs section defines the two input parameters and the values of the parameters are accessed via the ‘f_path’ intrinsic function.

The ‘f_path’ function is used to evaluate path expressions to access any data in your blueprint as well as any cloud resource data to which you have access. In our example, the path expression just uses the dot operator to access nested attributes, i.e. first access the Inputs attribute (i.e. the Inputs section) of the blueprint and within that, the UserId attribute, and within that the Value attribute.

Data Section (Named Literals)

Suppose your blueprint creates several databases and suppose that you don’t want to prompt the user for user and password. Furthermore, you want to code your blueprint so that it’s easy to change the password later.

In a procedural language, you’d use a named literal in order to document the intent and so that you can change it once at the top of your code. Within a blueprint, you do this by using the Data section.

```
Data:
  QADBCreds:
    user: sysman
    password: sysman
Resources:
  MyDB1:
    Type: application/oracle.com.cloud.common.DbPlatformInstance+json
    Container:
      f_getTemplateURI:
        - Small Database Service Template_automation_VIMAL_si
        - dbaas
    Properties:
      zone:
        f_getZoneURI:
          - f_path:
            - "Inputs.DbZone.Value"
          - dbaas
      params:
        username:
          f_path:
            - “Data.QADBCreds.user”
        password:
          f_path:
            - “Data.QADBCreds.password”
      name: jbName

  MyDB2:
    Container:
```

```
    ...
```

In the above case, you can see that the Data section takes a YAML structure, which can be traversed via the ‘path’ function, in the same way shown for Inputs earlier.

Putting It All Together – Multiple Interdependent Resources

In this more elaborate example, we show how one might create a database and an application that uses it. To do so, the blueprint specifies 4 cloud resources:

- Database service instance
- Java service instance
- Datasource of the Java service instance
- Application of the Java service instance

New constructs are in bold, explained below.

```
Inputs:
  DbZone:
    Type: String
    DefaultValue: Zone1
    Prompt: Zone to use for db
  DbPassword:
    Type: String
    DefaultValue: welcome1
    Prompt: Password to use for db
    Sensitive: True
Macros:
  # Return a name with unique (date-time) suffix
  # The one argument is a 'name' string
  f_myDescriptiveName:
    - 1
    - f_concat:
      - arg_1
      - '_'
      - f_path:
        - 'Info.time_suffix'
Resources:
  MyDB:
    Type: application/oracle.com.cloud.common.DbPlatformInstance+json
    Container:
      f_getTemplateURI:
        - Small Database Service Template_automation_VIMAL_si
        - dbaas
    Properties:
      zone:
        f_getZoneURI:
          - f_path:
            - "Inputs.DbZone.Value"
          - dbaas
      params:
        username: app_user
        password:
          f_path:
            - "Inputs.DbPassword.Value"
      name: jbName
  MyWebServer:
    Container:
      f_getTemplateURI:
        - PS4_LowHeapTemplate
        - jaas
    Properties:
      name:
        f_myDescriptiveName:
          - jb_pf
      zone:
        f_getZoneURI:
        - Zone1
        - jaas
  MyDS:
    Type: application/oracle.com.cloud.jaas.DataSource
    Container:
      f_getResourceURI:
        - MyWebServer
    Properties:
      name: QA_app_DS
      jndi_name:
        - jndi_1
        - jndi_2
      jdbc_driver: oracle.jdbc.OracleDriver
      database_type: Oracle
      database_connect_string:
         f_concat:
          - 'jdbc:oracle:thin:@'
          - f_getResourceAttr:
            - MyDB
            - connect_string
      username: app_user
      password:
          f_path:
           - "Inputs.DbPassword.Value"
  MyApp:
    Type: application/oracle.com.cloud.jaas.ApplicationInstanceDeployment
    Container:
      f_getResourceURI:
        - MyWebServer
    Properties:
      application_instance_component:
        f_getAppCompURI:
          - jbcomponent
          - SSA_USER1
          -
      name: myApp
Outputs:
  URL:
    Description: URL of the deployed app
    Value:
      f_getResourceAttr:
        - MyApp
        - http_application_invocation_url
```

Macro Section

If you have a sequence of constructs that you tend to repeat, you can use macro expansion to improve the readability of your blueprint. Macros also enable you to encapsulate logic e.g. so that you can need only modify the logic in one place to affect all code that refers to it.

Our example blueprint defines a macro named *f_myDescriptiveName.* It takes one string parameter and appends "_" as well as a string representation of the current time.

Attributes of Created Resources (Dependencies)

The key new feature introduced by this example is the ability to refer to attributes of created resources. For instance, the Cloud Resource Model for Datasource defines a database_connect_string attribute whose value is required to create a Datasource. The contents of the string won't be available until after the database is created, so it's clearly not something the blueprint author can know in advance. Instead, he uses an intrinsic function to refer to the needed property of the newly created database. (Each DatabasePlatformInstance exposes a 'connect_string' property.)

To do this, the Datasource specifies the value of JDBCConnectString via the 'f_getResourceAttr' intrinsic function:

```
MyDS:
  ...
  Properties:
     ...
    database_connect_string:
       f_concat:
        - 'jdbc:oracle:thin:@'
        - f_getResourceAttr:
          - MyDB
          - connect_string
     ...
```

In this case, the f_getResourceAttr function waits for the MyDB resource to be created and then returns the value of its connect_string property.[2]

Similarly, the creation of a Datasource is done by adding it to the JavaPlatformInstance that is created first, so we need to refer to the JavaPlatformInstance's URI.

```
MyDS:
  ...
  Container:
    f_getResourceURI:
      - MyWebServer
```

In this case, the f_getResourceURI function waits for the MyWebServer resource to be created and then returns its URI. (In addition to adding a Datasource to MyWebServer, we also must add an ApplicationInstanceDeployment, so the same approach is used for both.)

More generally, blueprint resources can refer to other resources and blueprint orchestration accounts for such dependencies, creating resources in parallel when possible.

---

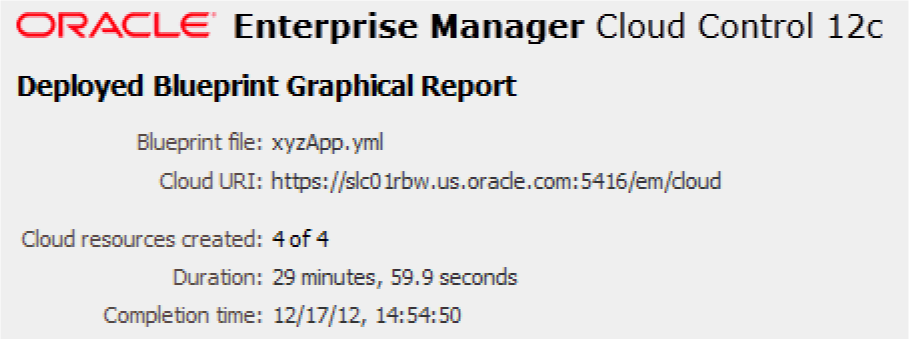
[2] You might think that f_path could be used to achieve the same effect, but only f_getResourceAttr knows to wait for the resource creation to succeed and its state be READY before attempting to get its attribute.

Visual Depiction of Blueprint Processing

The blueprint processor can also generate an HTML report that includes a graphical representation of the blueprint. This may be used to help understand the overall structure of the blueprint and the relationship of blueprint entities. In addition, the report can include the results of deploying the blueprint.

The following example report is for the example blueprint described in Processing a Blueprint.

The first part of the report summarizes the run:



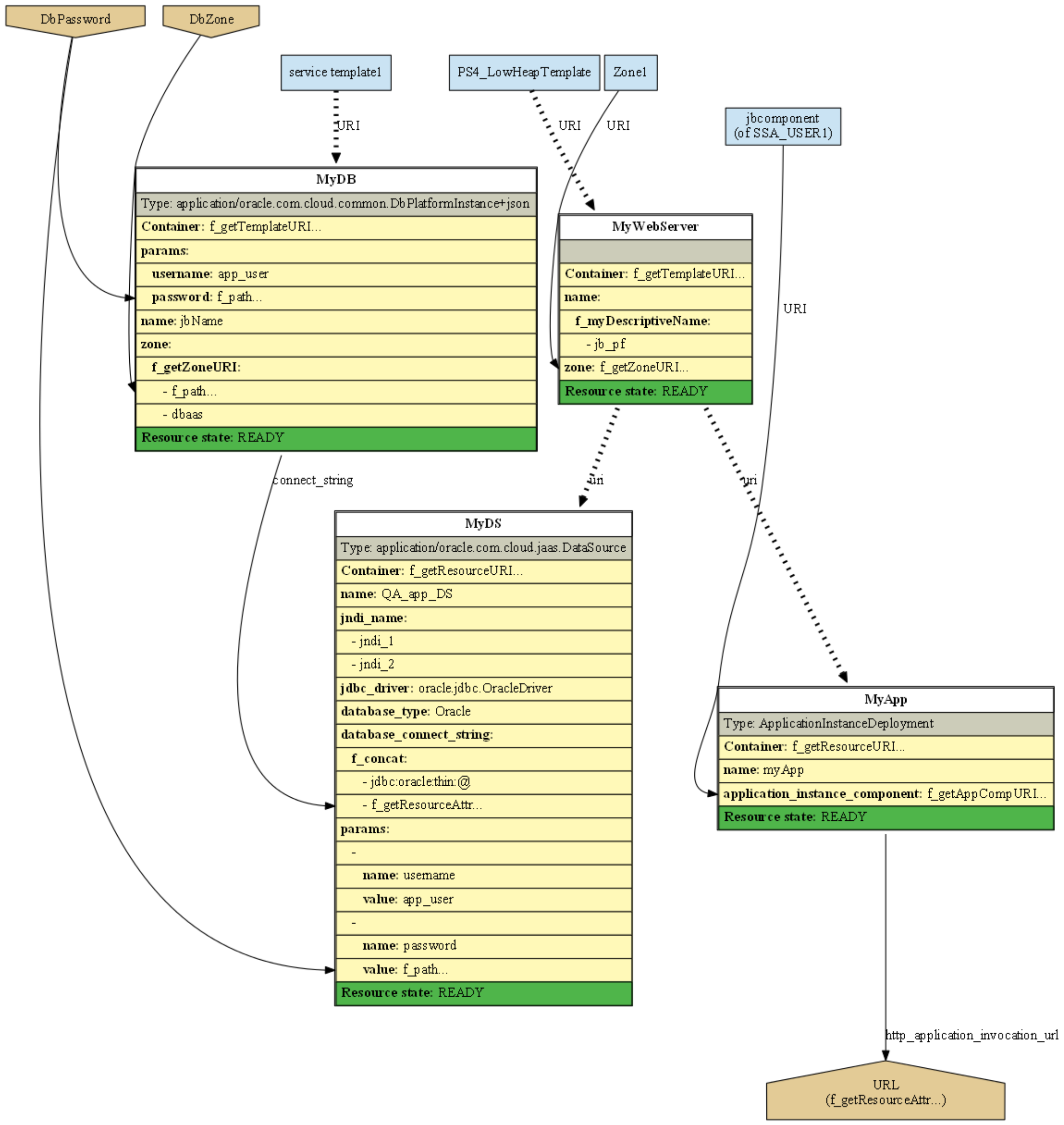
ORACLE Enterprise Manager Cloud Control 12c

Deployed Blueprint Graphical Report

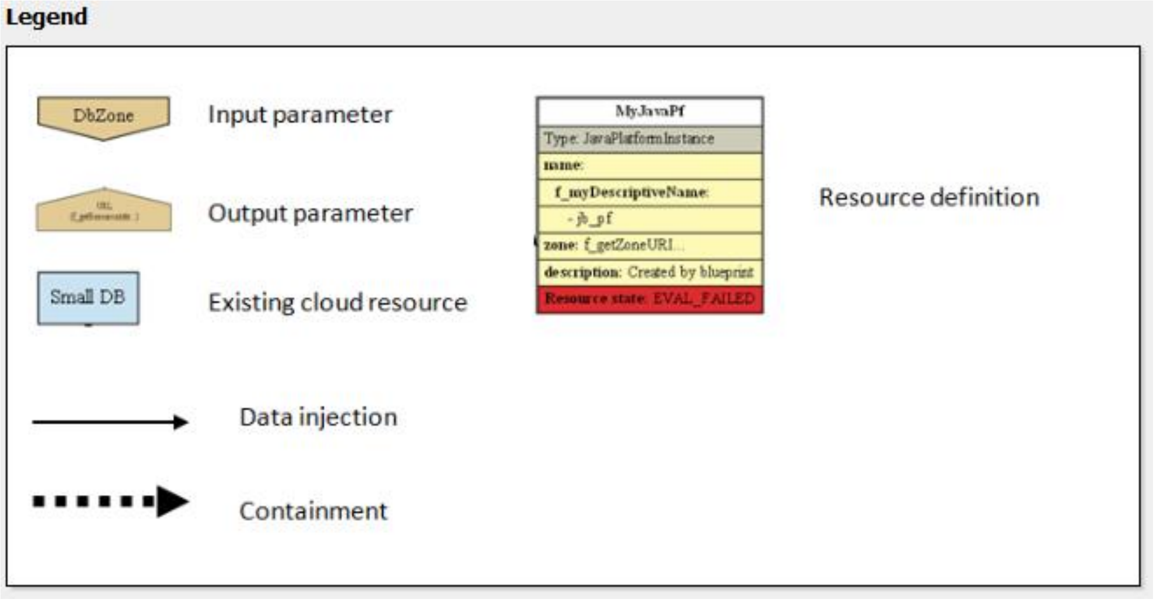
Blueprint file: xyzApp.yml
Cloud URI: https://slc01rbw.us.oracle.com:5416/em/cloud

Cloud resources created: 4 of 4
Duration: 29 minutes, 59.9 seconds
Completion time: 12/17/12, 14:54:50

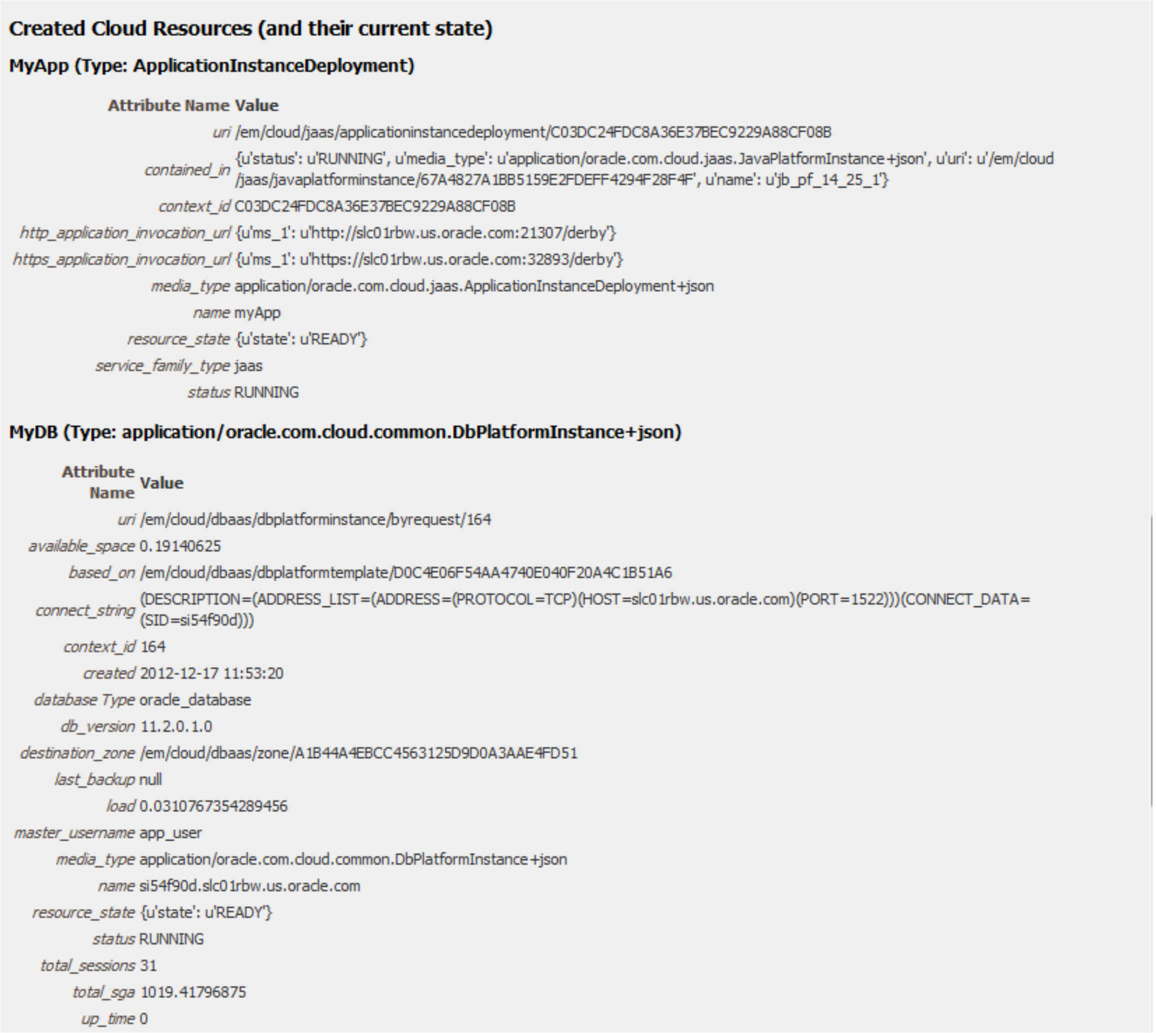
Next is the graphical depiction, in which two types of arcs are used. One depicts containment, e.g. containment of a datasource within a Weblogic server. The other depicts how data is used across the elements of a blueprint. If blueprint deployment is successful, the status of all resources will be Ready (green) and you can click on a resource, which links to another report section where you can view the values of its attributes at the time of creation



This is followed by a legend to explain the graphical conventions:



Finally, each created resource is summarized. (Clicking on a resource definition in the graphical depiction takes you directly to the resource summary for the selected resource.)



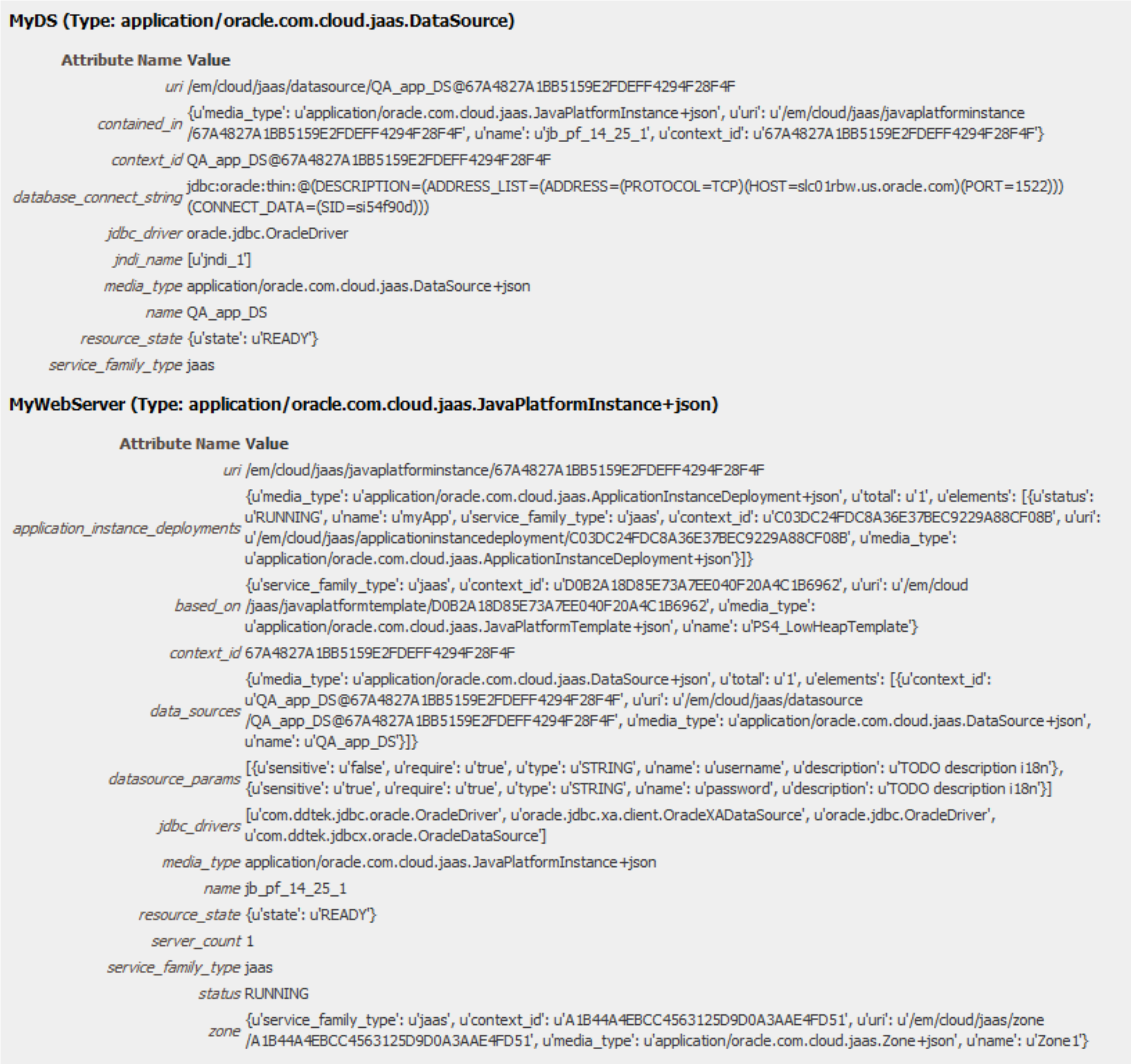
Created Cloud Resources (and their current state)

MyApp (Type: ApplicationInstanceDeployment)

Attribute Name	Value
uri	/em/cloud/jaas/applicationinstancedeployment/C03DC24FDC8A36E37BEC9229A88CF08B
contained_in	{u'status': u'RUNNING', u'media_type': u'application/oracle.com.cloud.jaas.JavaPlatformInstance+json', u'uri': u'/em/cloud/jaas/javaplatforminstance/67A4827A1BB5159E2FDEFF4294F28F4F', u'name': u'jb_pf_14_25_1'}
context_id	C03DC24FDC8A36E37BEC9229A88CF08B
http_application_invocation_url	{u'ms_1': u'http://slc01rbw.us.oracle.com:21307/derby'}
https_application_invocation_url	{u'ms_1': u'https://slc01rbw.us.oracle.com:32893/derby'}
media_type	application/oracle.com.cloud.jaas.ApplicationInstanceDeployment+json
name	myApp
resource_state	{u'state': u'READY'}
service_family_type	jaas
status	RUNNING

MyDB (Type: application/oracle.com.cloud.common.DbPlatformInstance+json)

Attribute Name	Value
uri	/em/cloud/dbaas/dbplatforminstance/byrequest/164
available_space	0.19140625
based_on	/em/cloud/dbaas/dbplatformtemplate/D0C4E06F54AA4740E040F20A4C1B51A6
connect_string	(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=slc01rbw.us.oracle.com)(PORT=1522)))(CONNECT_DATA=(SID=si54f90d)))
context_id	164
created	2012-12-17 11:53:20
database Type	oracle_database
db_version	11.2.0.1.0
destination_zone	/em/cloud/dbaas/zone/A1B44A4EBCC4563125D9D0A3AAE4FD51
last_backup	null
load	0.0310767354289456
master_username	app_user
media_type	application/oracle.com.cloud.common.DbPlatformInstance+json
name	si54f90d.slc01rbw.us.oracle.com
resource_state	{u'state': u'READY'}
status	RUNNING
total_sessions	31
total_sga	1019.41796875
up_time	0



MyDS (Type: application/oracle.com.cloud.jaas.DataSource)

Attribute Name	Value
uri	/em/cloud/jaas/datasource/QA_app_DS@67A4827A1BB5159E2FDEFF4294F28F4F
contained_in	{u'media_type': u'application/oracle.com.cloud.jaas.JavaPlatformInstance+json', u'uri': u'/em/cloud/jaas/javaplatforminstance/67A4827A1BB5159E2FDEFF4294F28F4F', u'name': u'jb_pf_14_25_1', u'context_id': u'67A4827A1BB5159E2FDEFF4294F28F4F'}
context_id	QA_app_DS@67A4827A1BB5159E2FDEFF4294F28F4F
database_connect_string	jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=slc01rbw.us.oracle.com)(PORT=1522)))(CONNECT_DATA=(SID=si54f90d)))
jdbc_driver	oracle.jdbc.OracleDriver
jndi_name	[u'jndi_1']
media_type	application/oracle.com.cloud.jaas.DataSource+json
name	QA_app_DS
resource_state	{u'state': u'READY'}
service_family_type	jaas

MyWebServer (Type: application/oracle.com.cloud.jaas.JavaPlatformInstance+json)

Attribute Name	Value
uri	/em/cloud/jaas/javaplatforminstance/67A4827A1BB5159E2FDEFF4294F28F4F
application_instance_deployments	{u'media_type': u'application/oracle.com.cloud.jaas.ApplicationInstanceDeployment+json', u'total': u'1', u'elements': [{u'status': u'RUNNING', u'name': u'myApp', u'service_family_type': u'jaas', u'context_id': u'C03DC24FDC8A36E37BEC9229A88CF08B', u'uri': u'/em/cloud/jaas/applicationinstancedeployment/C03DC24FDC8A36E37BEC9229A88CF08B', u'media_type': u'application/oracle.com.cloud.jaas.ApplicationInstanceDeployment+json'}]}
based_on	{u'service_family_type': u'jaas', u'context_id': u'D0B2A18D85E73A7EE040F20A4C1B6962', u'uri': u'/em/cloud/jaas/javaplatformtemplate/D0B2A18D85E73A7EE040F20A4C1B6962', u'media_type': u'application/oracle.com.cloud.jaas.JavaPlatformTemplate+json', u'name': u'PS4_LowHeapTemplate'}
context_id	67A4827A1BB5159E2FDEFF4294F28F4F
data_sources	{u'media_type': u'application/oracle.com.cloud.jaas.DataSource+json', u'total': u'1', u'elements': [{u'context_id': u'QA_app_DS@67A4827A1BB5159E2FDEFF4294F28F4F', u'uri': u'/em/cloud/jaas/datasource/QA_app_DS@67A4827A1BB5159E2FDEFF4294F28F4F', u'media_type': u'application/oracle.com.cloud.jaas.DataSource+json', u'name': u'QA_app_DS'}]}
datasource_params	[{u'sensitive': u'false', u'require': u'true', u'type': u'STRING', u'name': u'username', u'description': u'TODO description i18n'}, {u'sensitive': u'true', u'require': u'true', u'type': u'STRING', u'name': u'password', u'description': u'TODO description i18n'}]
jdbc_drivers	[u'com.ddtek.jdbc.oracle.OracleDriver', u'oracle.jdbc.xa.client.OracleXADataSource', u'oracle.jdbc.OracleDriver', u'com.ddtek.jdbcx.oracle.OracleDataSource']
media_type	application/oracle.com.cloud.jaas.JavaPlatformInstance+json
name	jb_pf_14_25_1
resource_state	{u'state': u'READY'}
server_count	1
service_family_type	jaas
status	RUNNING
zone	{u'service_family_type': u'jaas', u'context_id': u'A1B44A4EBCC4563125D9D0A3AAE4FD51', u'uri': u'/em/cloud/jaas/zone/A1B44A4EBCC4563125D9D0A3AAE4FD51', u'media_type': u'application/oracle.com.cloud.jaas.Zone+json', u'name': u'Zone1'}

Closing

This ends the introduction to cloud blueprints. Many features are only described in [BP_REF] including:

- Blueprint macros
- Path expressions for browsing the Cloud and blueprint
- Setting breakpoints and debugging blueprints
- Hints, tips, and frequently asked questions.

References

[BP_REF] ***Reference Manual - Cloud Blueprints and Blueprint Processor**,* included in the same directory in which this document was installed.

[CRMA] *Oracle Cloud Resource Model API;* The Cloud Resource Model API is described in ***Oracle® Enterprise Manager Cloud Administration Guide, 12.1.0.5, Part VII, Using the Cloud APIs***

[JSON] ***Introducing JSON**,* http://www.json.org/

[YAML] http://yaml.org/