blueplanet

Resource Adapter Development

Blue Planet DevOps Toolkit

Part 2 – Resource Providers

Rev 1.8

Blue Planet Resource Adapter (RA) Development

Agenda

1. Resource Providers
2. Onboarding to Blue Planet

blueplanet

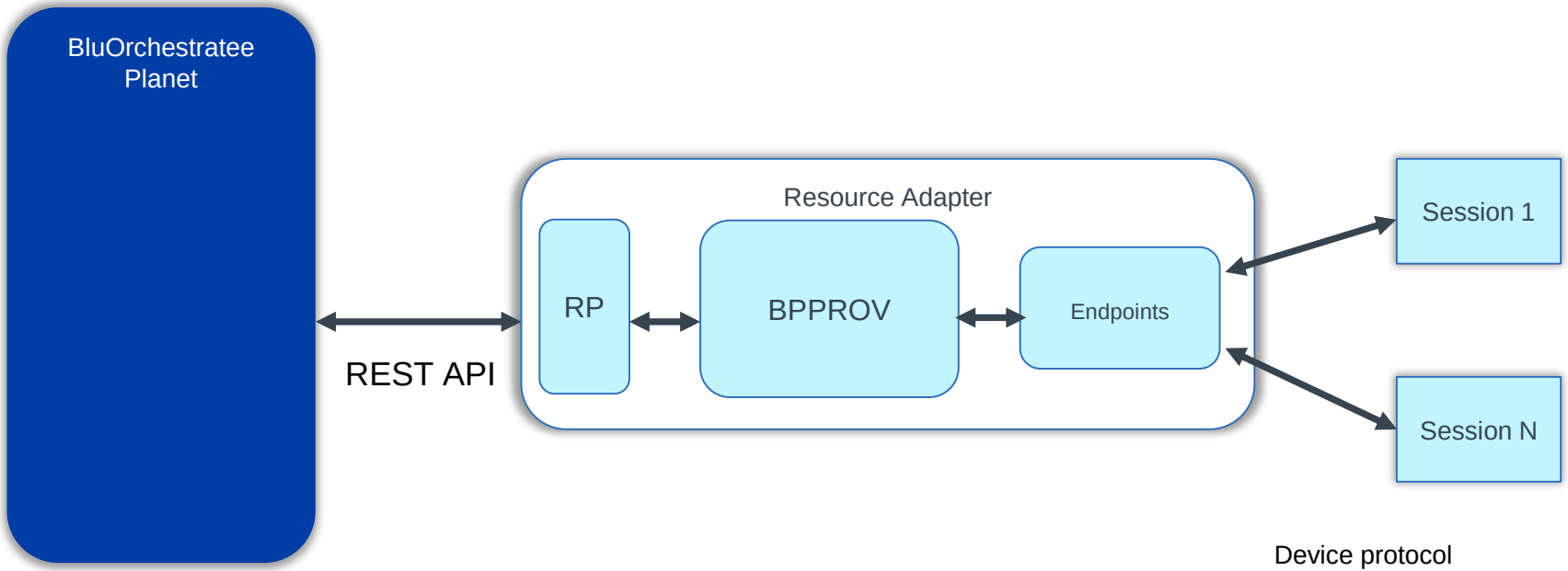
Resource Providers

Resource Providers - Preview

- Topics covered in this section include:
  - Overview of Resource Providers (RP)
  - Steps to configuring a RP
  - Installing RPSDK

blueplanet

Resource Adapter Architecture



Creating a Resource Provider

1. Verify rpsdk is installed
2. Configure Resource Types
3. Map commands
4. Create Sessions
5. Orchestrate Onboarding

blueplanet

Install RPSDK

- The **paster** command adds rpsdk to **setup.py**

```
…
install_requires=[
    'rasdk','rpsdk[ra]'
    ],
…
```

- If you remove rpsdk or don't install the RP when using the **`paster`** command:
  - Remove **env** directory
  - Re-run **`make`** command
  - Update **requirements.txt** by running **`make requirements`**
- **DO NOT** re-run **`paster`** command

blueplanet

Lab: Adding RPSDK

- Complete the following lab:
  - Lab 20: Adding RPSDK

Optional lab provided for reference only - do not complete unless your instructor indicates you should.

blueplanet

Review – Key BPO Concepts

Key BPO Concepts - Preview

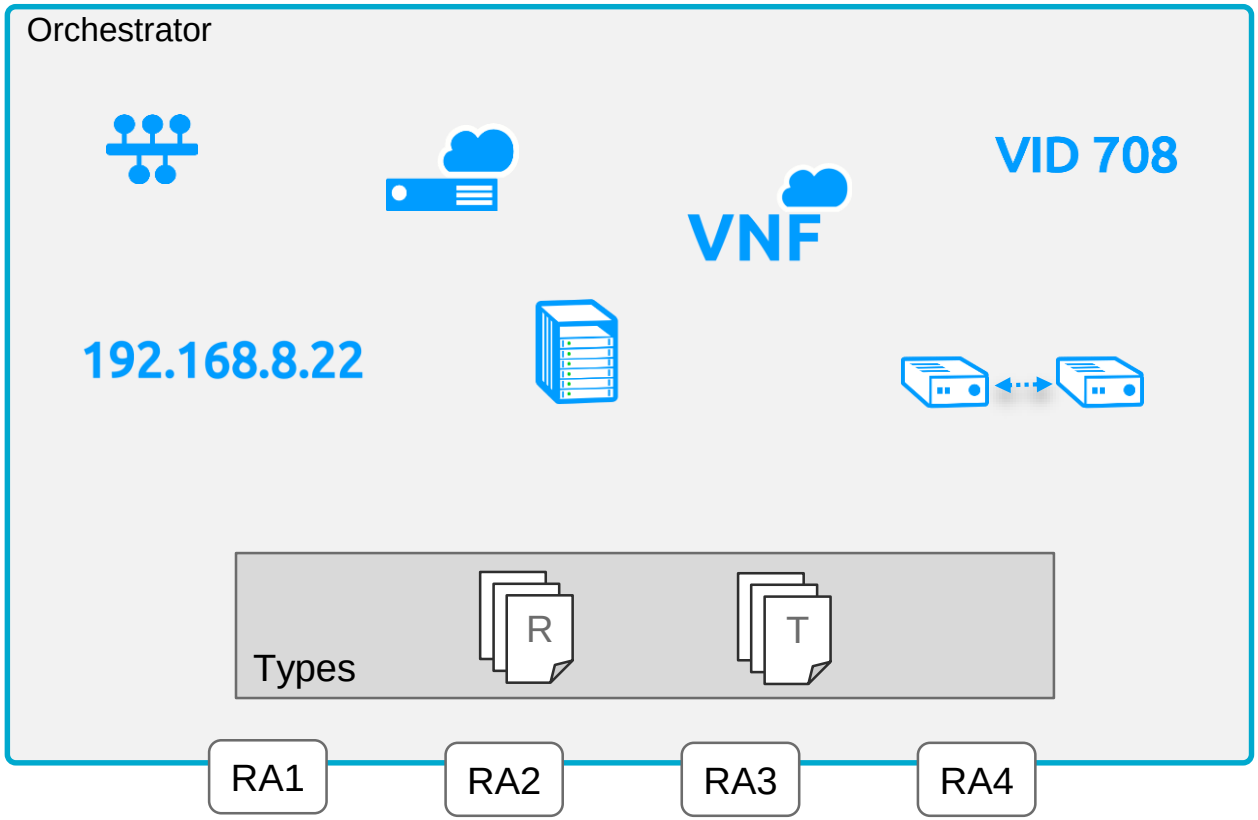
- Topics covered in this section include:
  - Resource types
  - Products
  - Resources

blueplanet

Resource Management

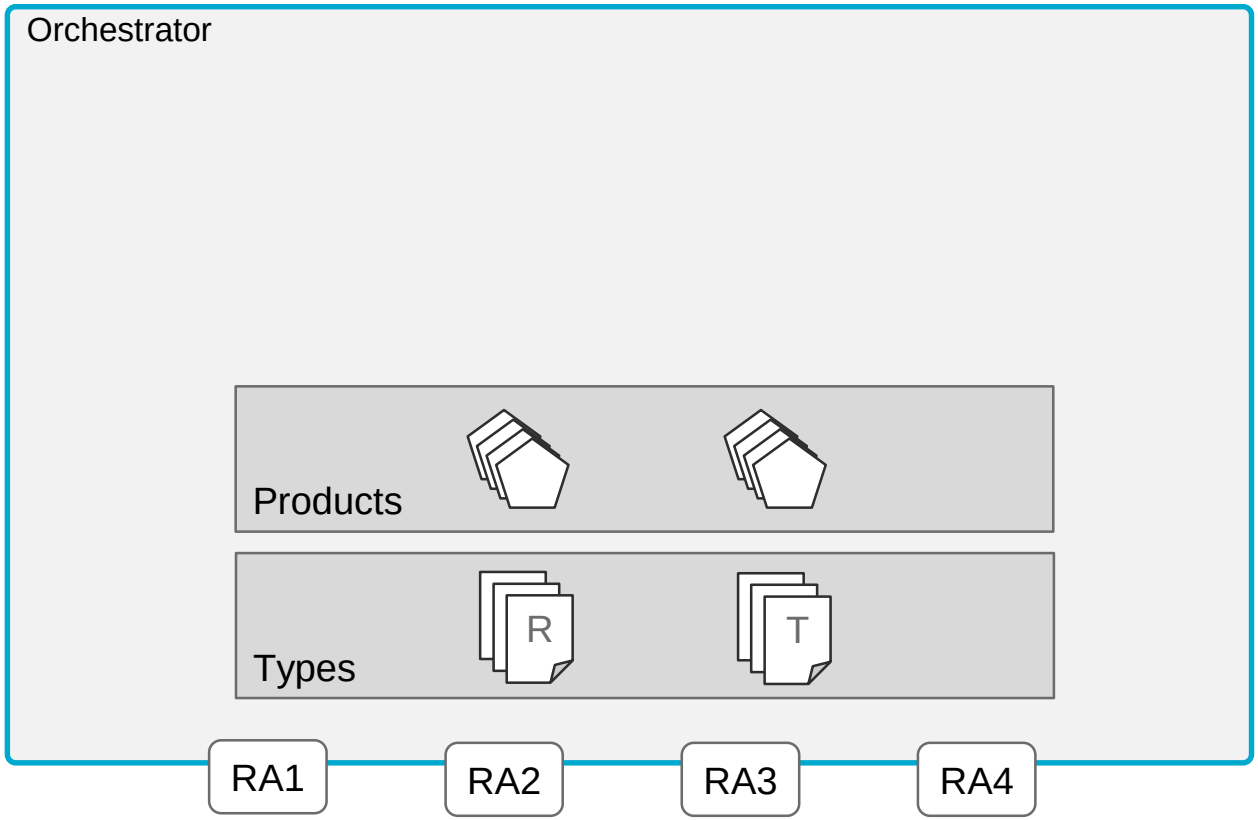
blueplanet

Type Catalog



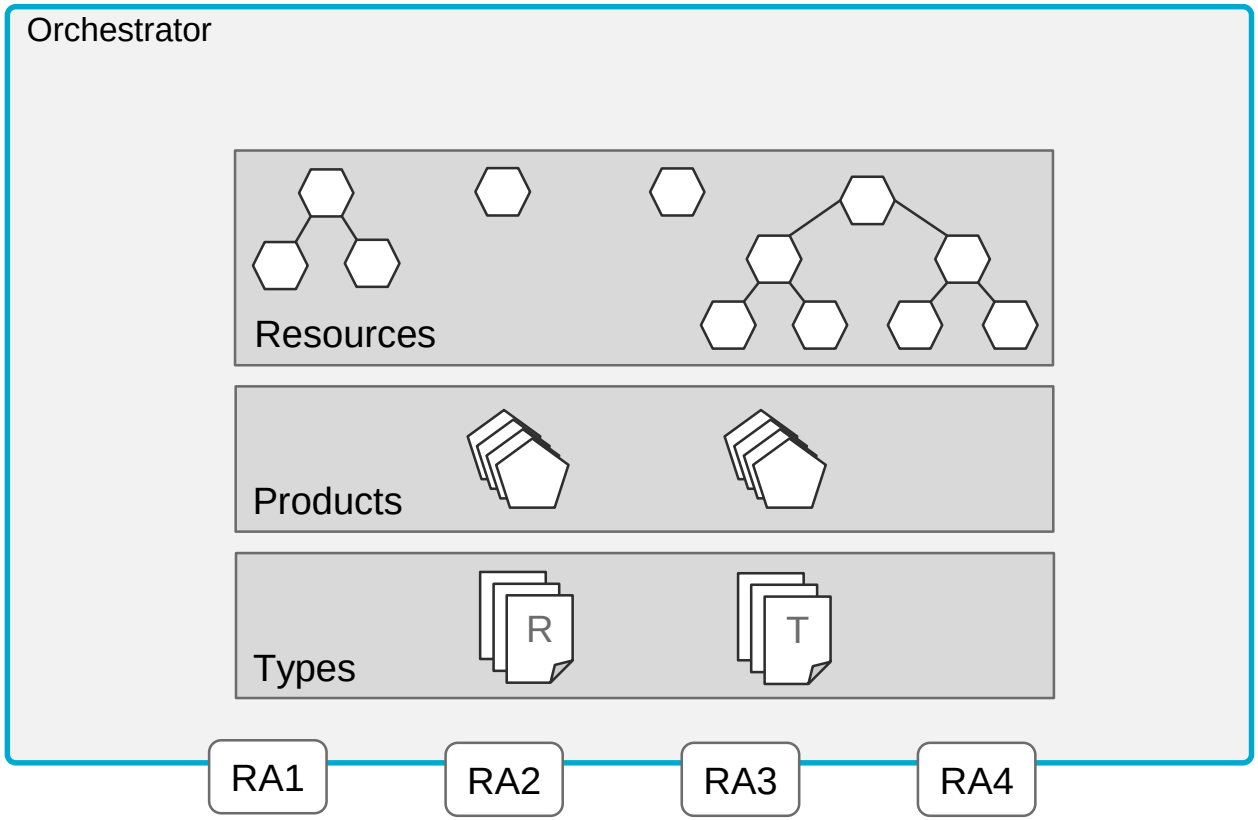
blueplanet

Products



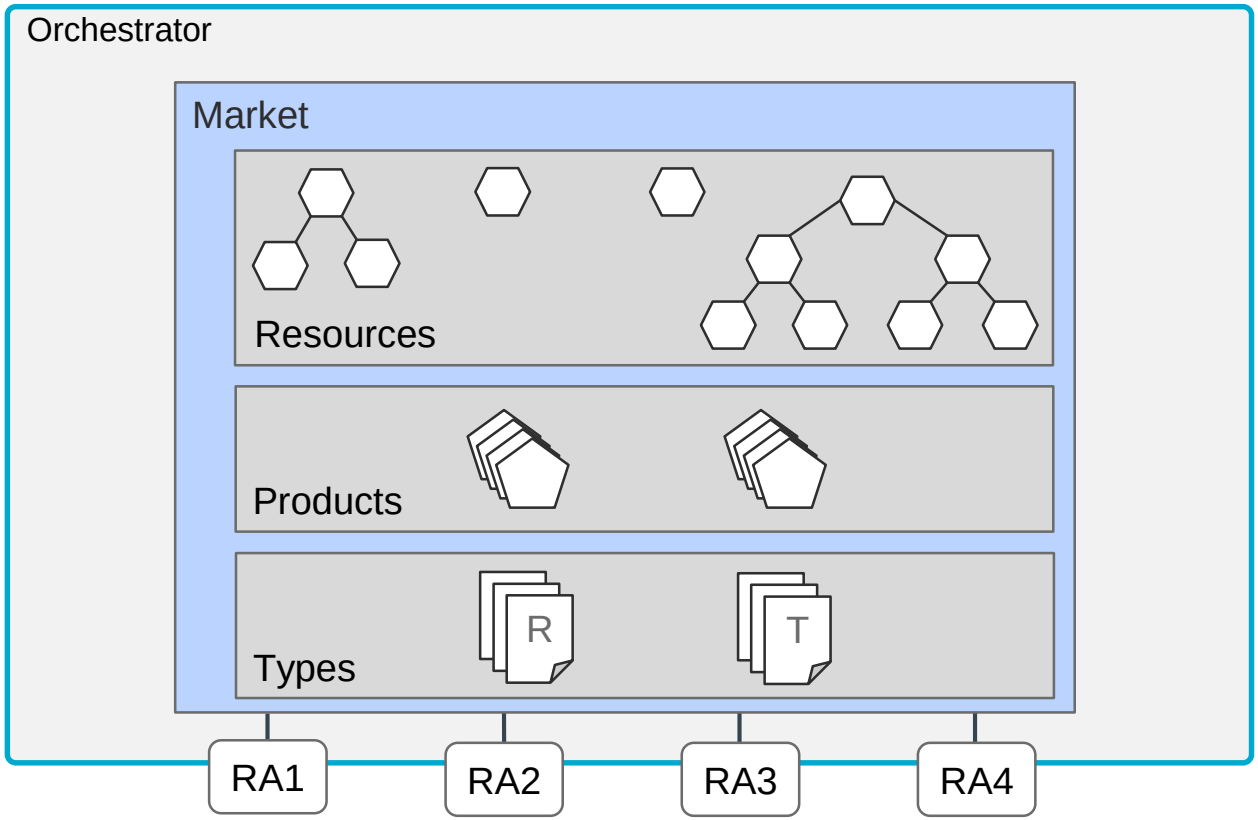
blueplanet

Resources



blueplanet

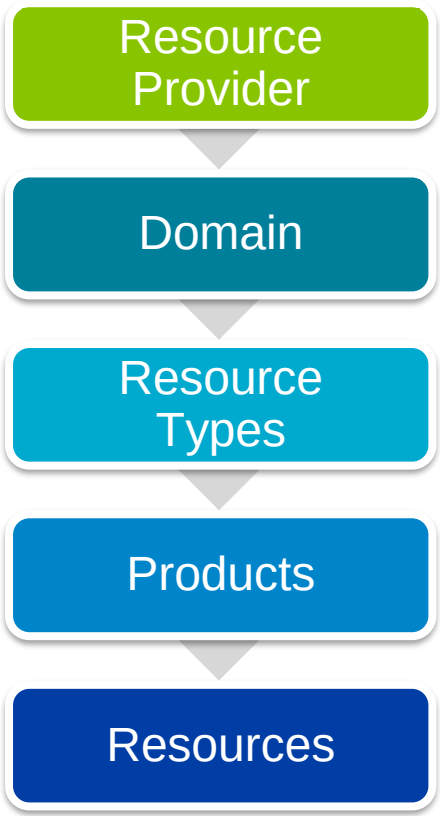
Market



blueplanet

Resource Management

- Products link Resources to a Domain
  - Each Domain is linked to one Resource Provider
  - Domains expose Resource Types and Products
  - Each Product is linked to exactly one Domain via the Resource Type.
  - Products link *Resources* to a *Resource Type*
    - Each Resource is linked to one Domain
    - Each Resource is linked to exactly one Product.



blueplanet
a division of ciena

Model Definitions

Model Definitions - Preview

- Topics covered in this section include:
  - The settings.py file parameters for Resource Providers
  - Resource Adapter types
  - The rp_config.yaml file
  - Overview of Resource Definitions
  - Resource Discovery Strategies
  - Defining Products

blueplanet

Create Model Definitions

- High-level process:
  1. Update **settings.py** (if needed)
  2. Create resource type definition files
  3. Add resource types to **rp_config.yaml**
  4. Define relationships
  5. Create **products.json**
  6. Create UI Schema

blueplanet

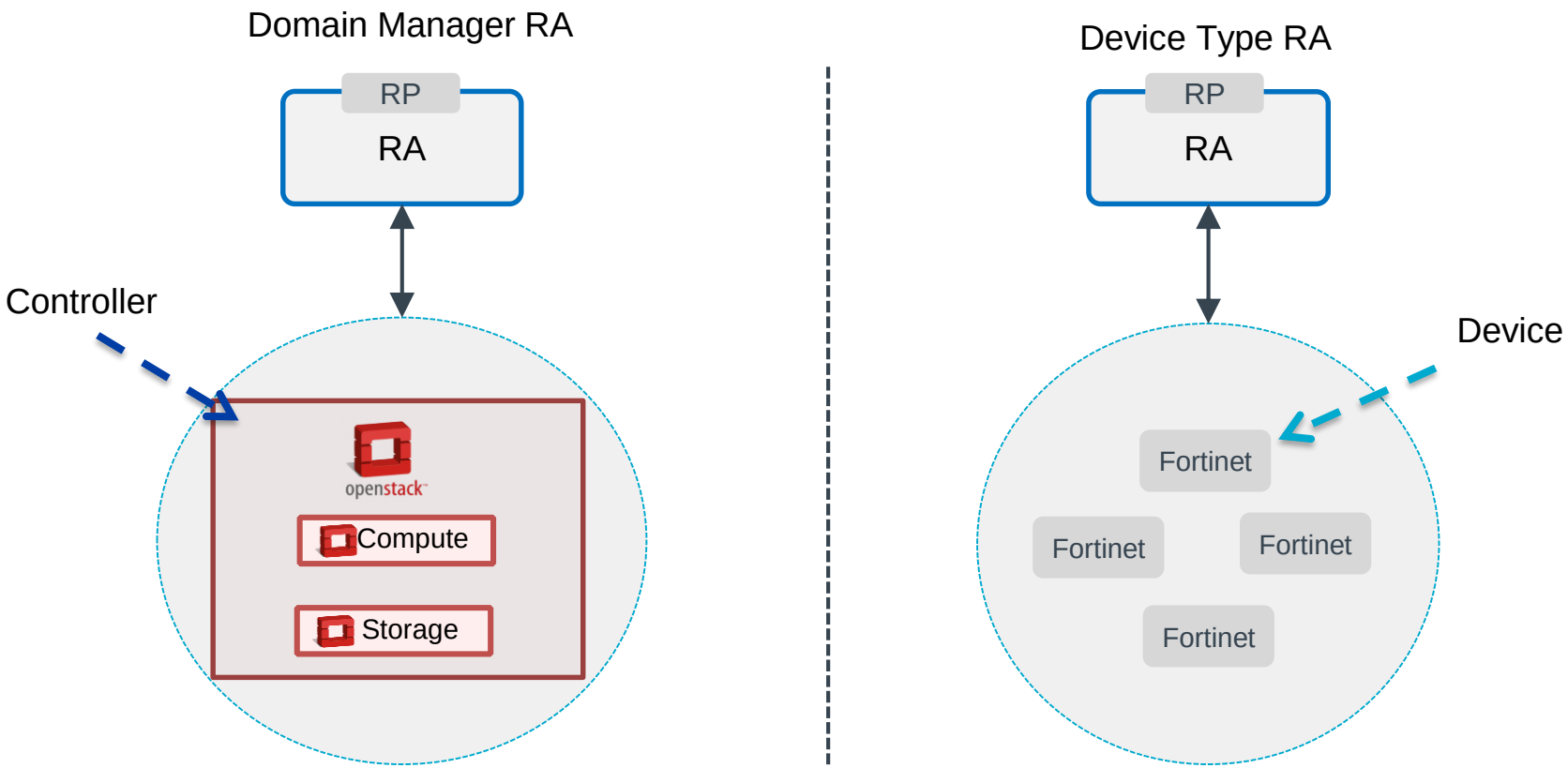
settings.py

- RP_CONFIG
  - Path to the **rp_config.yaml** file.
- RESOURCES_DIR
  - Directory where the RP configuration files are located.
- DOMAIN_MANAGER_RP
  - Boolean – whether the RP is a domain manager or not.

```
RESOURCES_DIR = os.path.join(BPPROV_MODEL_DIRPATH, 'resources')
RP_CONFIG = os.path.join(RESOURCES_DIR, 'rp_config.yaml')
DOMAIN_MANAGER_RP = True
```

- The **`paster`** utility configures these settings, but you can change them if needed.

Resource Adapters Types



blueplanet

Resource Adapters Types - details

- The primary difference between these two types of RAs is how session management is done:
  - For device type RAs, sessions are tied to the resources - each resource type that your RA defines needs to have a property for holding the device session id
  - For device type RAs, you need a **resource type** for each device you are managing that contains all the connection and session information
  - In the domain manager type RA, there is one session per domain - all of the resources know about which session they belong to because they know which domain they belong to
  - For the domain manager type RA you do not need to define the resource type for the device you are connecting to, but you can if it is desired

Resource Adapters Types – docker RA

- The RA that was created in previous labs is a CLI RA that talks to an Operating System (Ubuntu)
- The OS then tells docker to launch containers
- As a result, this CLI RA is a domain manager type RA as it is using another domain (docker) to configure a device (the docker container)

rp_config.yaml

- Required file
- Create by **`paster`** if RP option is set
- Defines domain settings
- Defines “resource types” exposed to Orchestrate
  - Note that “resource types” for BPO are not the same as RESOURCE_TYPES for the RA!
- Must be placed in directory specified in settings.py
  - Typically **model/resources/rp_config.yaml**
- The **`paster`** creates a base **rp_config.yaml** file

Resource Definitions

- All definitions are based on Blue Planet Orchestrate JSON schemas
- Written in HOCON
  - Text file with **.tosca** extension
- Defines:
  - Properties
  - Capabilities
  - Requirements
- Resource definitions go in location defined in **rp_config.yaml**
  - Location is relative to the location of **rp_config.yaml**
  - Typically **model/resources/definitions**
- Additional information: Attend C-802 class or Reference Documentation -> Definition Modules -> Resource Types in https://community.ciena.com/docs/DOC-1881

Definition File Format

Definition Metadata

Definition

```
1  "$schema"    = "http://cyaninc.com/json-schemas/tosca-lite-v1/definition-module#"
2  title        = "vFirewall resource type"
3  package      = tosca
4  version      = "1.0"
5  description = "This document defines the VirtualFirewall resource type."
6  authors      = [ "training@cyaninc.com" ]
7
8  imports {
9    Vnf = tosca.resourceTypes.VirtualNetworkFunction
10 }
11
12 resourceTypes {
13
14   VirtualFirewall {
15
16     derivedFrom = Vnf
17     title = Virtual Firewall
18     description = """
19       A Firewall appliance as a VNF
20     """
21
22     properties {
23
24       publicNet {
25         title = "Public network"
26         description = "UUID of the public network where the Firewall shall be accessible"
```

Package name

Resource name

Resource Type Identifier = *package*.resourceTypes.*resourceName*

blueplanet

Add Resource Types to rp_config.yaml

- All resources types must be in **rp_config.yaml**
- Include resource_types under **domain** section
- **id** must be resource type identifier

```
resource_types:
       - id: "package.resourceTypes.resourceName1"
         discoveryStrategy:
           - strategyType: "sync"
             pollingMode: "listOnly"
             pollingSettings:
               - pollingIntervalMs: 30000
       - id: "package.resourceTypes.resourceName2"
         discoveryStrategy:
           - strategyType: "sync"
             pollingMode: "listOnly"
             pollingSettings:
               - pollingIntervalMs: 30000
```

Resource Discovery Strategies

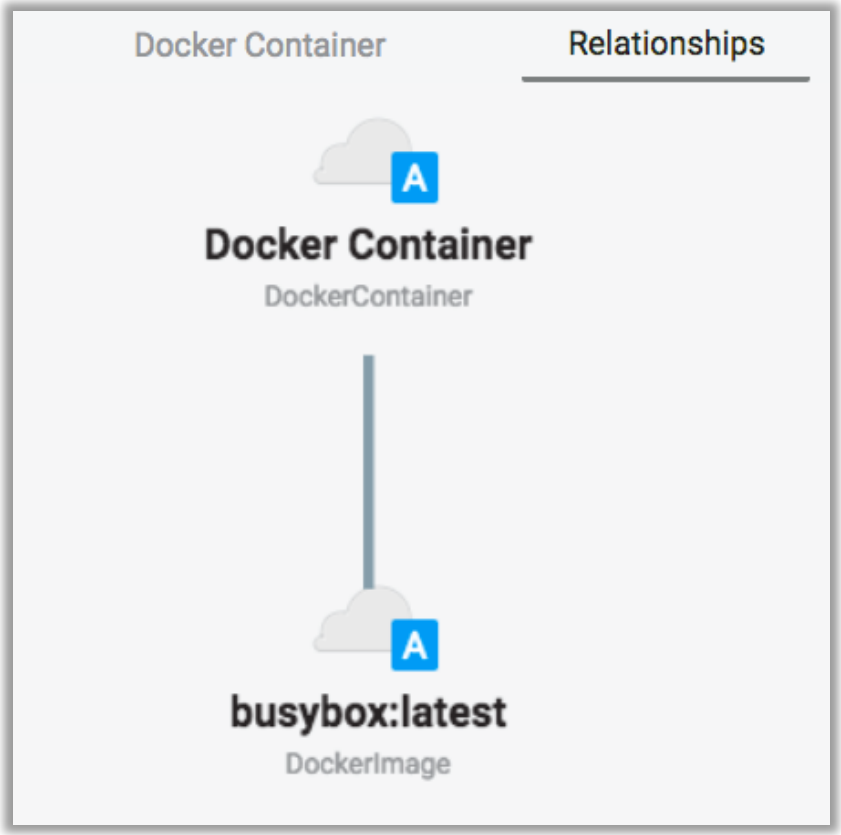
- Set per resource type
- Set in strategyType field
  - **sync –** RA Manager will do periodic polling to determine the status of resources, and update Orchestrate Market database
  - **async –** RA Manager will not do any polling but relies on the RA to send asynchronous updates
- Sync pollingMode:
  - **ListOnly –** Discover new resources, update existing resources, remove deleted resources
  - **GetOnly –** Update existing resources or remove deleted resources
- Resource discovery happens after domain is created in Orchestrate

Define Relationships

- Add relationships to **rp_config.yaml**

```
relationships:
  - targetTypeId: "docker.resourceTypes.DockerImage”
    sourceTypeId: "docker.resourceTypes.DockerContainer"
    capabilityName: "containers"
    requirementName: "image"
    relationshipTypeId: "tosca.relationshipTypes.DependsOn"
    fieldKind: "provider"
    fieldLocator: "image"
```

Define Relationships



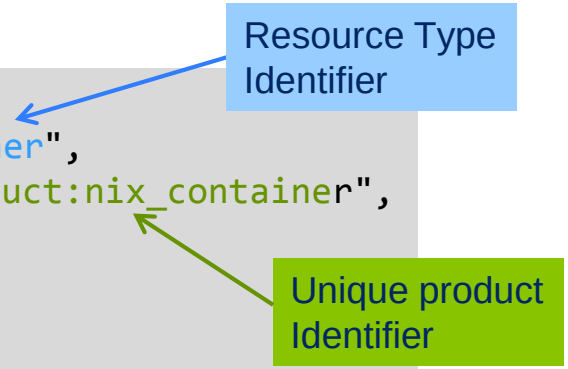
blueplanet

Define Products

- Any resource type that will be orchestrated requires a product
  - Product name must be unique for the Resource Adapter
  - Active products are available to users in the Web UI
  - Inactive products are only available to Blue Planet Orchestrator (BPO)
  - RPSDK only supports a single product per resource type
  - Advanced products (Service Template-based) are covered in the C-802 class and are manually created and uploaded into BPO
  - provider_data is extra data provided by the RA to describe the product (normally used for Service Template association)
- **products.json** defines “products” to be onboarded to Orchestrate
  - Save in **model/resources**

Define Products - Example

```
[{
    "resource_type": "docker.resourceTypes.nixContainer",
    "provider_product_id": "urn:cienatraining:bp:product:nix_container",
    "title": "Docker Container",
    "active": true,
    "provider_data": {}
}]
```



blueplanet

blueplanet
a division of ciena

UI Schemas

UI Schemas - Preview

- Topics covered in this section include:
  - UI Schema 1 vs UI Schema 2
  - The create.json, edit.json and details.json
  - UI Schema 2 directory structure

blueplanet

UI Schema Versions

UI Schema 1.x

- 16.06 and earlier
- Single JSON file for a product
- UI for creating resource instances
- Use with orchestrate-ui-dev and DevOps Toolkit
- Details in Appendix A

UI Schema 2

- 16.10 and later
- Utilizes Frost- Ciena’s open-source UI development framework.
- Supports data-driven UI views
- Not implemented in DevOps Toolkit 16.10

UI Schema 2 - Create, Edit, Detail

- UI Schema is optional
- Must be valid JSON
  - Use http://jsonlint.com/ to validate JSON

Schema Files

create.json

- Specifies the UI Schema when creating the resource type
- All properties including those that will not be updateable

edit.json

- Specifies the UI Schema when editing the resource type
- Only properties that can be updated via the UI

details.json

- Specifies how the UI Schema is displayed when viewing the resource type

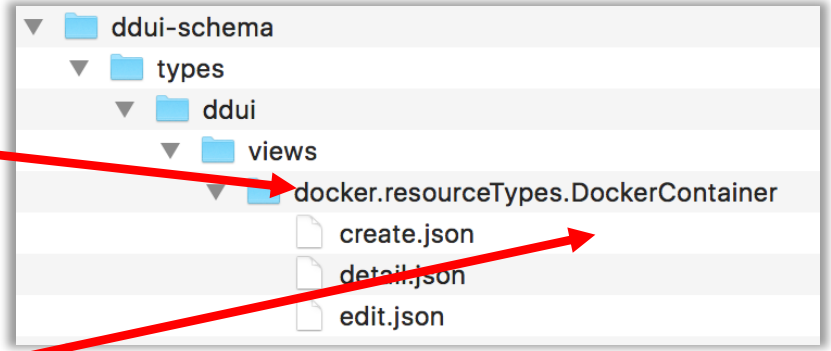
blueplanet

UI Schema 2 - Directory Structure

```
"$schema" = "http://cyaninc.com/json-schemas/to
  title = "Docker container resource"
  package = docker
  version = "1.0"
  description = "This document defines DockerCo
  authors = ["RA Developer (ra.developer@ciena.

  imports {
  Root = tosca.resourceTypes.Root
  Container = tosca.capabilityTypes.Container
  }

  resourceTypes {
    DockerContainer {
      derivedFrom = Root
      title = "Container"
      description = """
      A docker container
   """"
```



blueplanet

UI Schema 2 - Format

Version: always set to “2.0”

Type: “form” or “details”

List of tab elements

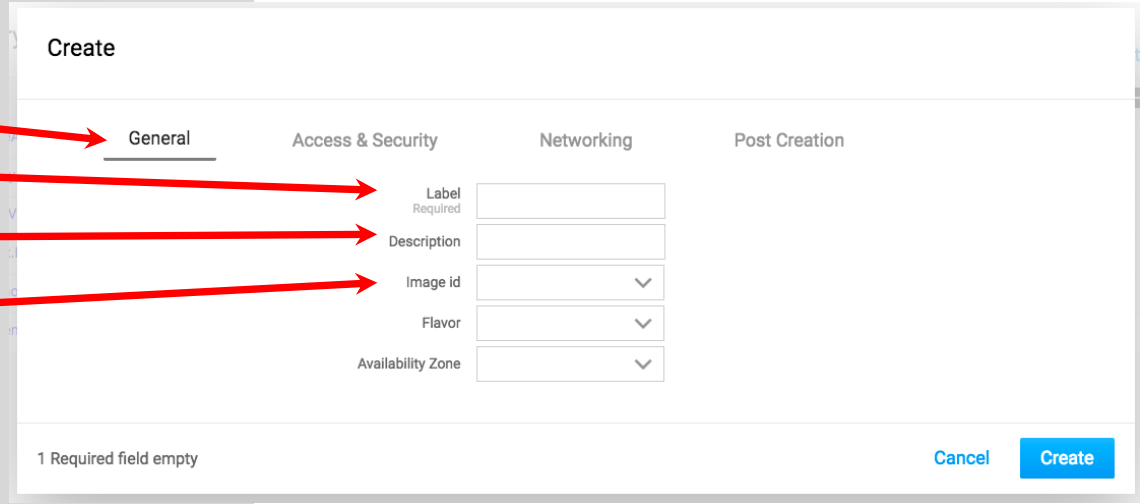
List of items within each tab

```
{
    “version”: “2.0”,
    “type”: ”form”,
    “cells”: [{
        “label”: “label name”,
        “children”: [{
            “model”: “label”
          },{
            “model”: “”,
              “renderer”: {
                “name”: “”,
                ”options”: {
                  “modelType”: “”,
                    ”query”: {
                      “resourceTypeId”: “
                    }
                  }
            }
          }],
```


blueplanet

UI Schema 2 - Edit Form

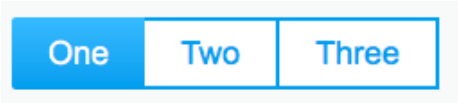
```
"version": "2.0",
   "cells": [{
     "label": "General”
     "children": [{
        "model": "label"
     }, {
        "model": "description”
      }, {
        "model": "properties.imageID",
        "renderer": {
          "name": "select",
           "options": {
              "modelType": "resource",
             "query": {
                "resourceTypeId": "tosca.resourceTypes.Image”
              }
           }
        } }, {
        "model": "properties.flavorID",
        "renderer": {
          "name": "select",
           “options” {
```



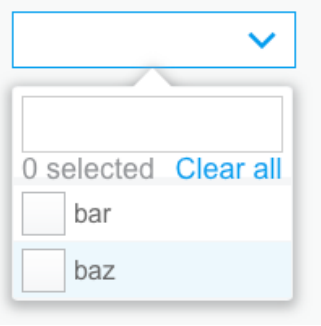
blueplanet

UI Schema 2 - Renderer

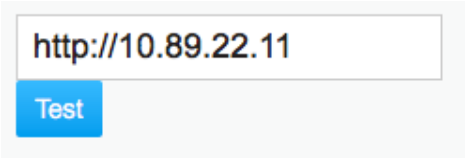
Renderers
boolean
button-group
checkbox-array
geolocation
image
multi-select
number
password
property-chooser
select
static
text
url



button-group

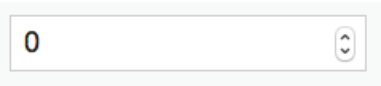


multi-select



url

Boolean



number

text

blueplanet

Lab Create Model Definitions

- Complete the following lab:
  - Lab 21: Create Model Definitions

Notify your instructor when you have completed these labs.

blueplanet

blueplanet
a division of ciena

Command mapping
&
Session mapping

Command mapping & Session mapping - Preview

- Topics covered in this section include:
  - Command mapping process
  - The commands.json file
  - CRUDL mapping
  - Creating session mapping

blueplanet

Command Mapping

- **commands.json**
  - Links each product to a command mapping file that lists all CRUDL commands

```
{
  "urn:cienatraining:bp:product:nix_container": "container.json",
  "urn:cienatraining:bp:product:nix_image": "image.json"
}
```

- Note: This file is not necessary if the commands are stored in a file named after the resource type:
  - Example: test.resourceTypes.set = **test.resourceTypes.set.json**

Blue Planet Equivalent for CRUDL

- Create - post
- Retrieve - get
- Update - put
- Delete - delete
- List - list

Command Mapping File

Described in Appendix B

Product ID from **products.json**

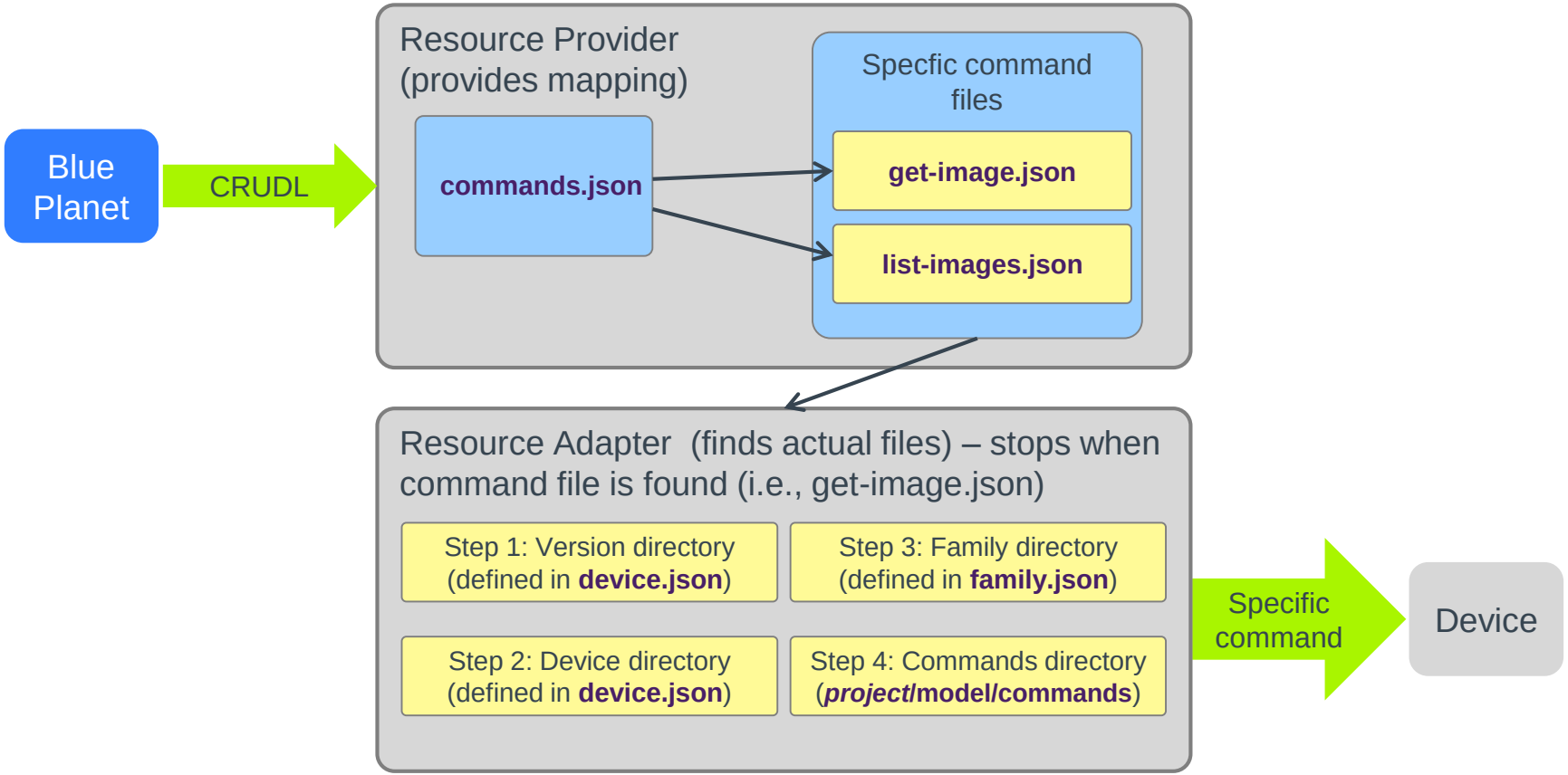
Resource Type Identifier from **rp_config.yaml**

Map Orchestrate CRUDL commands to bpprov commands

```
{
    "type": "bpprov_full",
    "product": "urn:ciena:bp:product:docker_image",
    "resource": "docker.resourceTypes.DockerImage",
    "provider_resource_id_field": "tag",
    "commands": {
        "get": {
            "command": "get-image.json"
        },
        "list": {
            "command": "list-images.json"
        }
    }
}
```


blueplanet

The Big Picture



blueplanet

Lab: Command Mapping

- Complete the following lab:
  - Lab 22: Command Mapping

Notify your instructor when you have completed these labs.

blueplanet

Creating Sessions

- Session mapping file
  - Save in **model/resources**
  - Default file (**session-templates.json**) was created by the **`paster`** utility
  - Points RP to template with session configuration information
  - Example:

```
"urn:ciena:bp:domain:radocker":{
        "template": "ra-session-create.tmpl"
    }
```

Matches the “domains” id setting in **rp_config.yaml**

```
domains:
  - id: "urn:ciena:bp:domain:radocker"
```

- Session template file
  - Jinja template based on session POST schema
    - http://192.168.33.10:8080/api/v1/swaggerui/#!/sessions/post
  - Save in **model/templates**

Session template file

- Example:

```
{
    "authentication": {
        "cli": {
            "username": "{{ username }}",
            "password": "{{ password }}“
        }
    },
    "connection": {
        "hostname": "{{ hostname }}",
        "cli": {
            "hostport": {{ hostport|int or 22 }}
        }
    },
    "description": "{{ description }}",
    "typeGroup": "/typeGroups/NixServer",
    "id": "{{ domainId }}"
}
```

Values that will be provided by user when creating domain in the Web UI

NixServer needs to be changed to match the TYPE_GROUP setting in the RA’s **settings.py** file

This value is derived from the Session Mapping file (**session-templates.json**)

blueplanet

Lab: Create Session Templates

- Complete the following lab:
  - Lab 23: Create Session Templates

Notify your instructor when you have completed these labs.

blueplanet

blueplanet
a division of ciena

Onboarding to Blue Planet

Onboarding to Blue Planet - Preview

- Topics covered in this section include:
  - Steps to onboard RA on Devops TK
  - Steps to onboard RA on production BP server

Devops TK Onboarding Overview

Process:

1. Set Environment Variables
2. Add bpocore-dev IP to **known_hosts** file
3. Run RA in dev mode

- bpocore-dev and orchestrate-ui-dev docker containers must be running before starting RA

blueplanet

RA Onboarding

- With RP configured, RP will automatically connect to Orchestrate
- Resources types will be automatically onboarded
- Domain still needs to be created
  - Requires Domain UI-Schema
  - UI-Schema tells Orchestrate what data to collect when adding the domain
- For Vagrant environment:
  - RA runs in virtualenv
  - No hooks to BP platform
  - You must specify environment variables so RP can find bpocore-dev
  - Must add bpocore-dev IP address to **~/.ssh/known_hosts**
- Use `source /home/vagrant/examples/example-ra-cisco/scripts/orch-env`

DevOps Environment Variables

- BPOCORE_IP – IP address of bpocore-dev docker container
- DOCKER_BRIDGE_IP – docker bridge IP
- MARKET_URL – URL to Market API
- ASSETS_URL – URL to Asset Manager API

Run Your RA

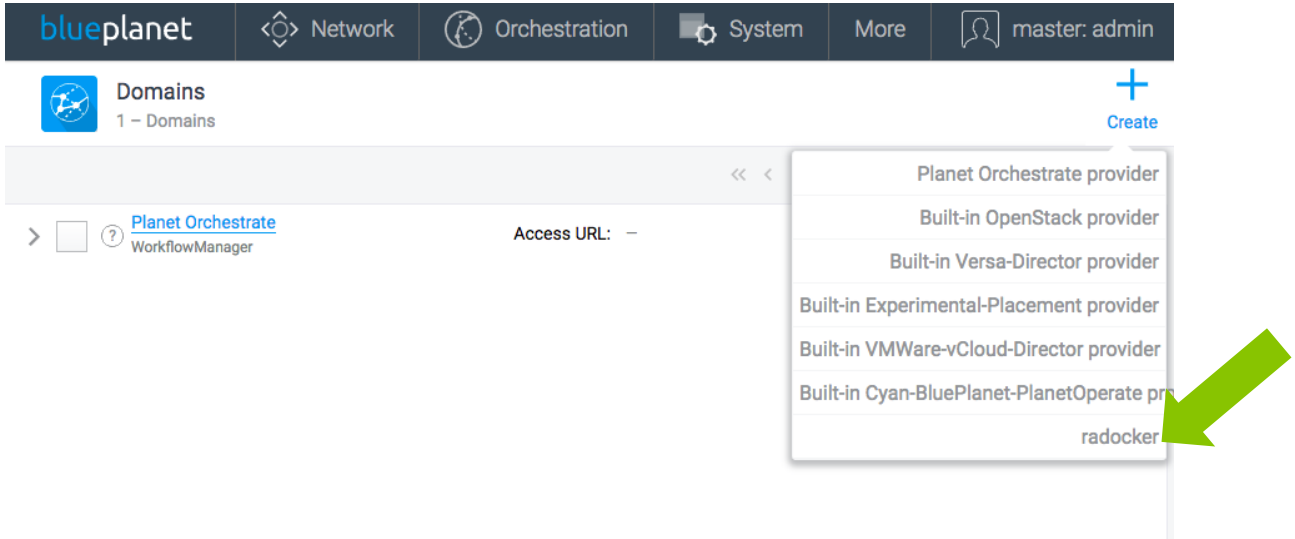
```
env/bin/radocker --dev --declare-rp
```

- Starts RA in dev mode – Not in a docker container
- Enables RP

blueplanet

Create Domain

- Go to Orchestrate UI – http://192.168.33.10:9980/orchestrate
- Use Domains screen (Orchestration → Domains) to add a new domain
- Click Create to display the RP list



Lab: Onboard Your RA to Orchestrate

- Complete the following lab:
  - Lab 24: Onboard Your RA to Orchestrate

Notify your instructor when you have completed these labs.

blueplanet

Orchestrate Docker Containers

- ResourceTypes and Products are automatically onboarded
- View products and discovered resources under the domain
  - Products: There should be two
  - Resources: bpocore-dev and orchestrate-ui-dev should be listed
- Your Container product will be available under **All Products**
  - Create a new docker container

blueplanet

Lab: Start and Stop a Docker Container

- Complete the following lab:
  - Lab 25: Start and Stop a Docker Container

Notify your instructor when you have completed these labs.

Onboarding to a Blue Planet Server Overview

Process:

1. Build a Docker image
2. Build a solution
3. Move the solution to Blue Planet server

blueplanet

Build a Docker image

- Make sure RA is built (`make prepare-venv`)
- Edit the user.email and user.name attributes in the **Dockerfile** with your information:

```
git config --global user.email "radocker@ciena.com" && \
git config --global user.name "radocker"
```

- Run the `make image` command
- Verify image by running the `docker images` command
  - You should see cyan/*yourRA* and cyan/*yourRA*-base

blueplanet

Build a solution

- Copy the **fig.yml** file from the **~/examples/example-ra-cisco** directory to the top-level project directory:

```
$ cp ~examples/example-ra-cisco/fig.yml .
```

- Modify the **fig.yml** file to describe your RA image
- Build the solution with solmaker:

```
$ solmaker build fig.yml
```

- Verify image by running the `docker images` command
  - You should see cyan/*solution-example*

```
#fig.yml file
__version__: 1
solution_name: example
solution_version: 0.0.1
apps:
    example:
        image: cyan/racisco:latest
        volumes:
            - /dev/log:/dev/log
            -
/etc/hostname:/etc/physical_hostname:ro
```


blueplanet

Move the solution to Blue Planet server

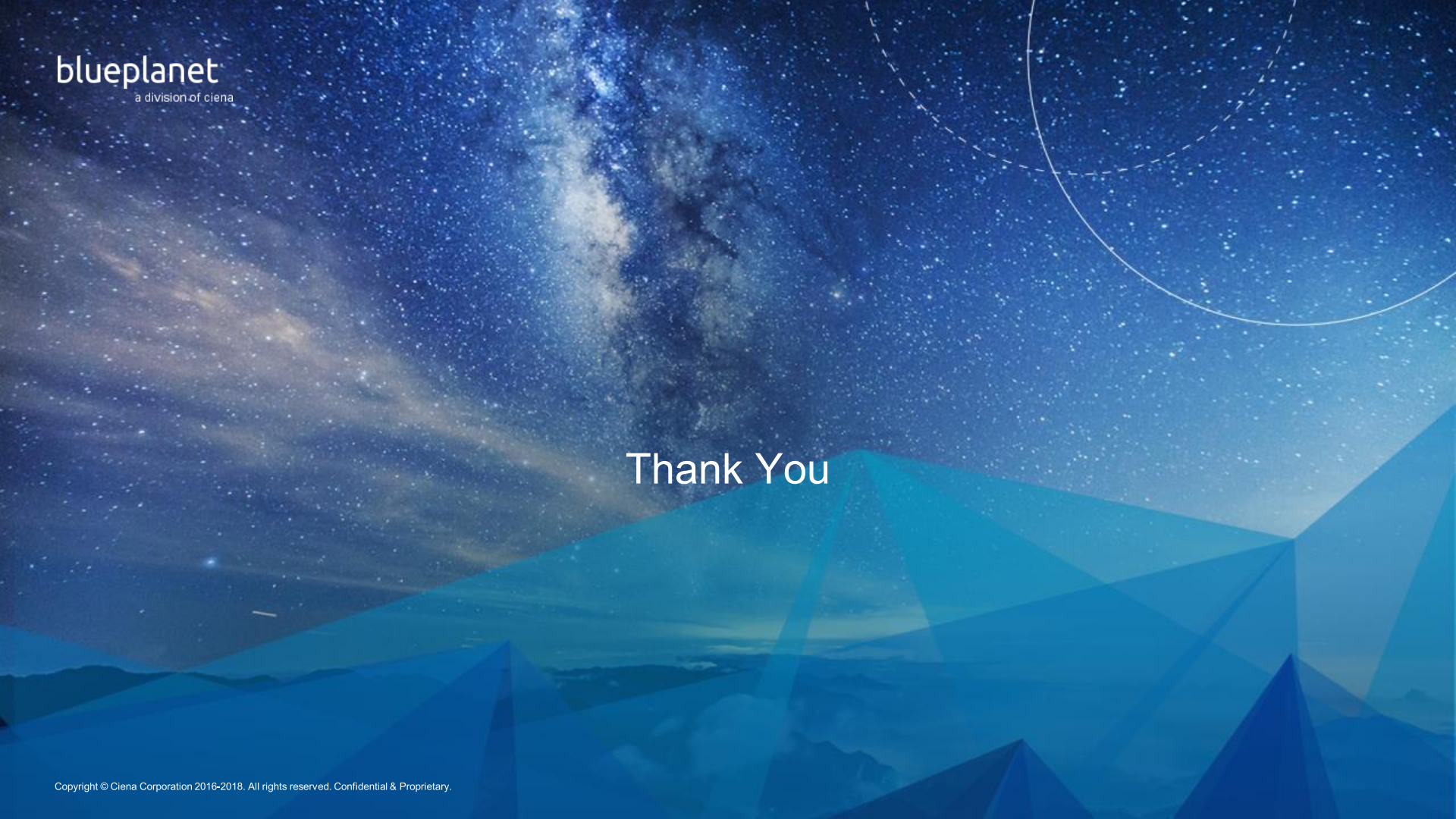
- Execute the `did-save` command to save the solution image
- Follow the prompts and choose the cyan/*solution-example* image that you created
- Generate an RSA key in the **~/.ssh** folder:

```
$ ssh-keygen -t rsa -C "your_email@yourcompany.com"
```

- Add the public key to the list of known hosts in the destination server (in **~/.ssh/known_hosts**)
- Push the solution to a Full Blue Planet server:

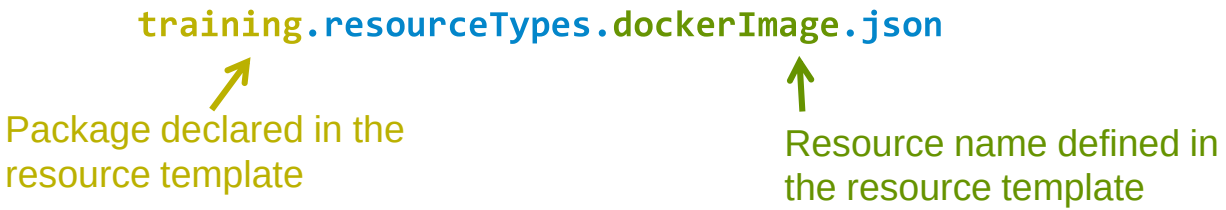
```
$ did-push --dest user@ip:did
```

- Load the transferred image on the new server with the `did-load` command.



Appendix A - UI Schema 1 Files

- User Interface (UI) schema file(s)
  - The file name must have the following parts delineated by '.' characters:



  - Must be valid JSON
    - Use http://jsonlint.com/ to validate JSON
- UI Schema go in location defined in rp_config.yaml
  - Location is relative to the location of rp_config.yaml
  - Typically models/resources/ui-schema

UI Schema 1 File Format

```
{
    "resourceTypeIdentifier" : {
        "fields" : {
},
        "fieldSets": {
            "fields" : {
            }
        }
        "fieldGroups": {
            "fields": {
            },
            "fieldSets": {
                    "fields" : {
                }
            }
        }
    }
}
```


blueplanet

Appendix B - Type values

- The type value specifies the **driver factory**, which is used to indicate how commands are run on the RA:

device	• Session bearing resources in a device-manager RA.
bpprov	• Resources using simplified bpprov command execution.
bpprov_full	• Resources using bpprov command execution.
playbook	• Resources using playbook.
bpprov_exec	• One-time command push to a device in a device-manager RA.