blueplanet

Resource Adapter Development

Blue Planet DevOps Toolkit 17.06.1

Part 1 – Resource Adapters

Rev 1.8

Blue Planet Resource Adapter (RA) Development

Agenda

1. Blue Planet and RA Development Overview
2. Creating an RA Project
3. bpprov Overview
4. Creating a CLI-Based RA
5. Customizing an RA

blueplanet

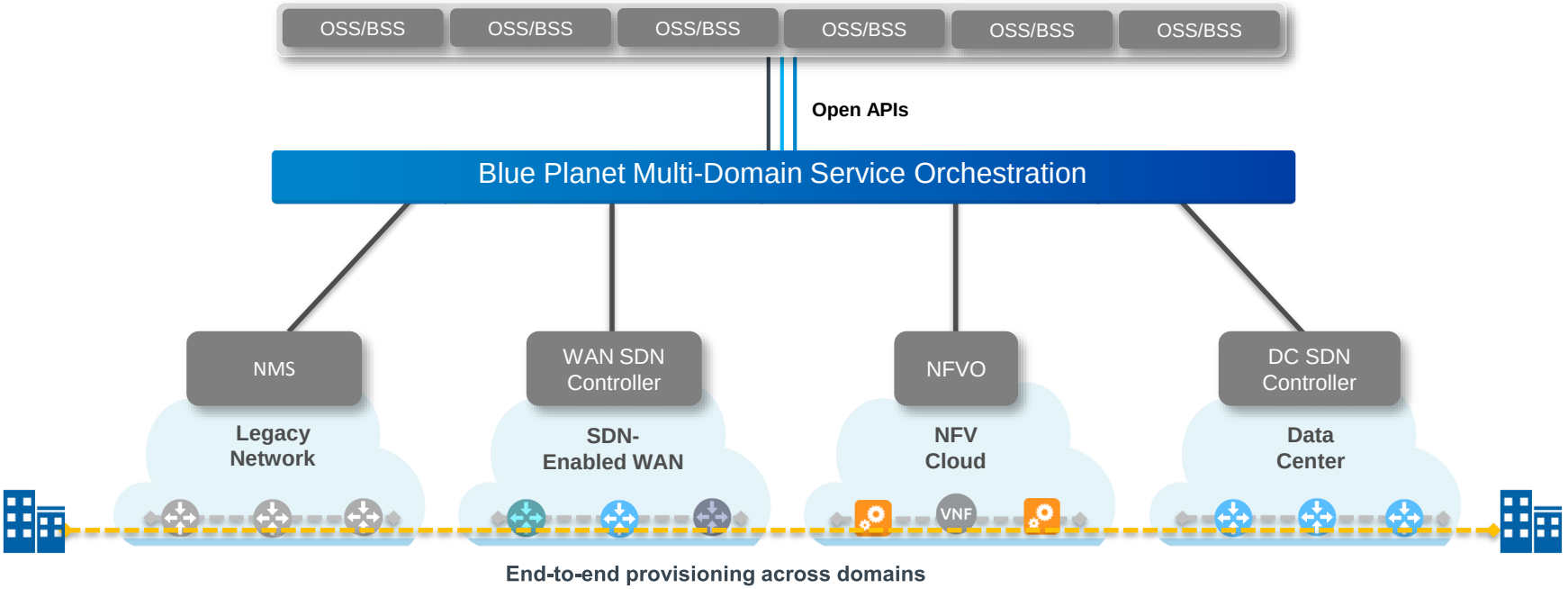
blueplanet
a division of ciena

Blue Planet Architecture

Blue Planet Architecture - Preview

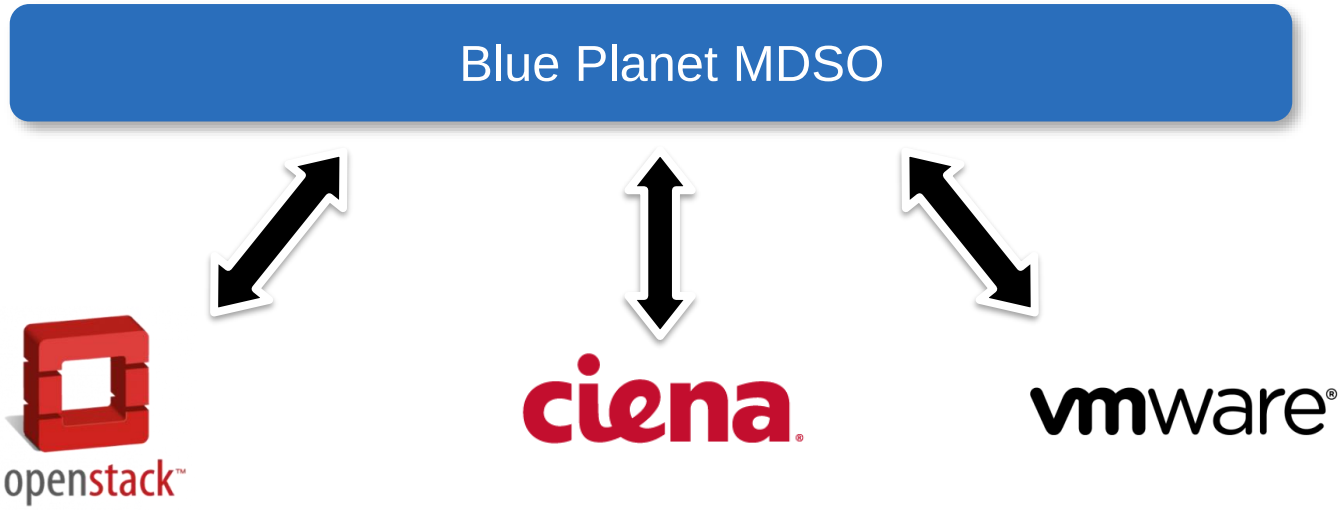
- Topics covered in this section include:
  - Overview of Blue Planet MDSO
  - Key terms, such as Domain and Resource Adapter
  - Introduction to Micro-Services, Blue Planet Orchestration and Resource Adapter Architectures
  - Blue Planet Developers Exchange
  - The Devops TK
  - Review of docker and key docker commands

Blue Planet Big Picture



blueplanet

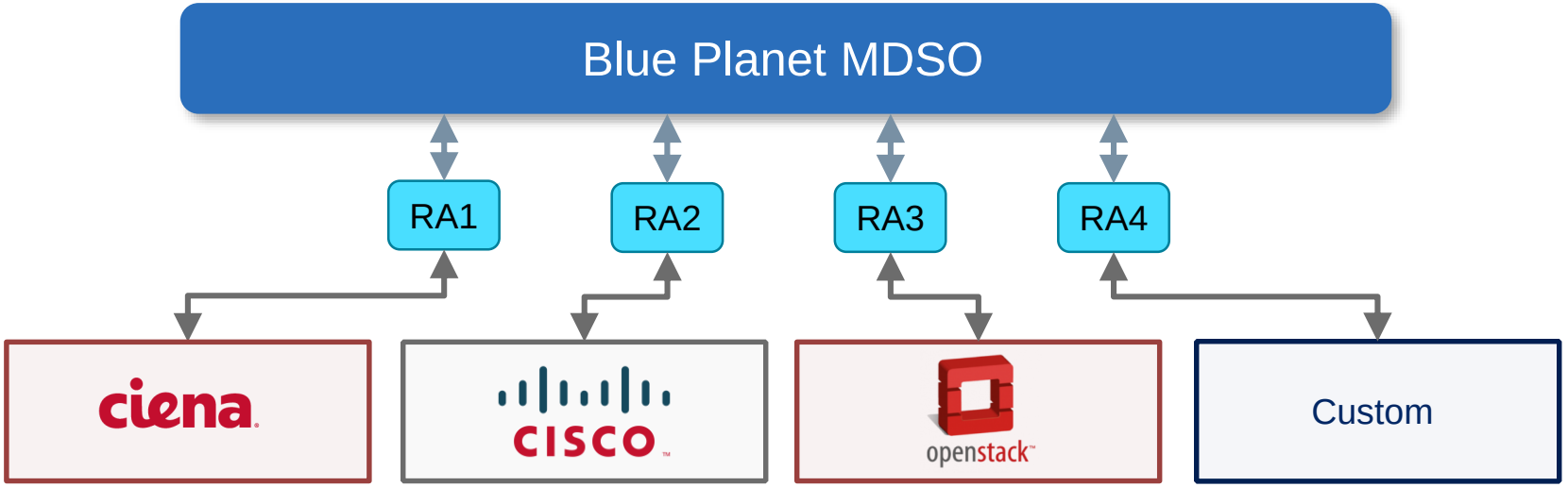
Domains



Domain:

A group of similar physical and logical resources under a single administrative umbrella

Resource Adapters



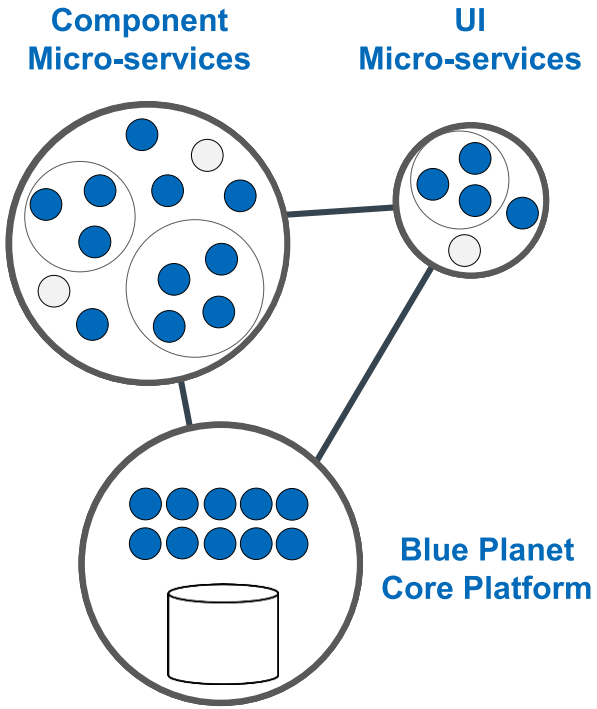
Resource Adapter

Microservice running on the Blue Planet server that communicates with a specific type of device

Micro-Services Architecture

- Tailored suites of micro-services enable Blue Planet to support different use cases
  - SDN Management and Control
  - NFV Service Orchestration
  - Multi-Domain Service Orchestration
  - Or, any mix of use cases

- Ala-carte customization enables a single platform to meet customer's business and operational requirements

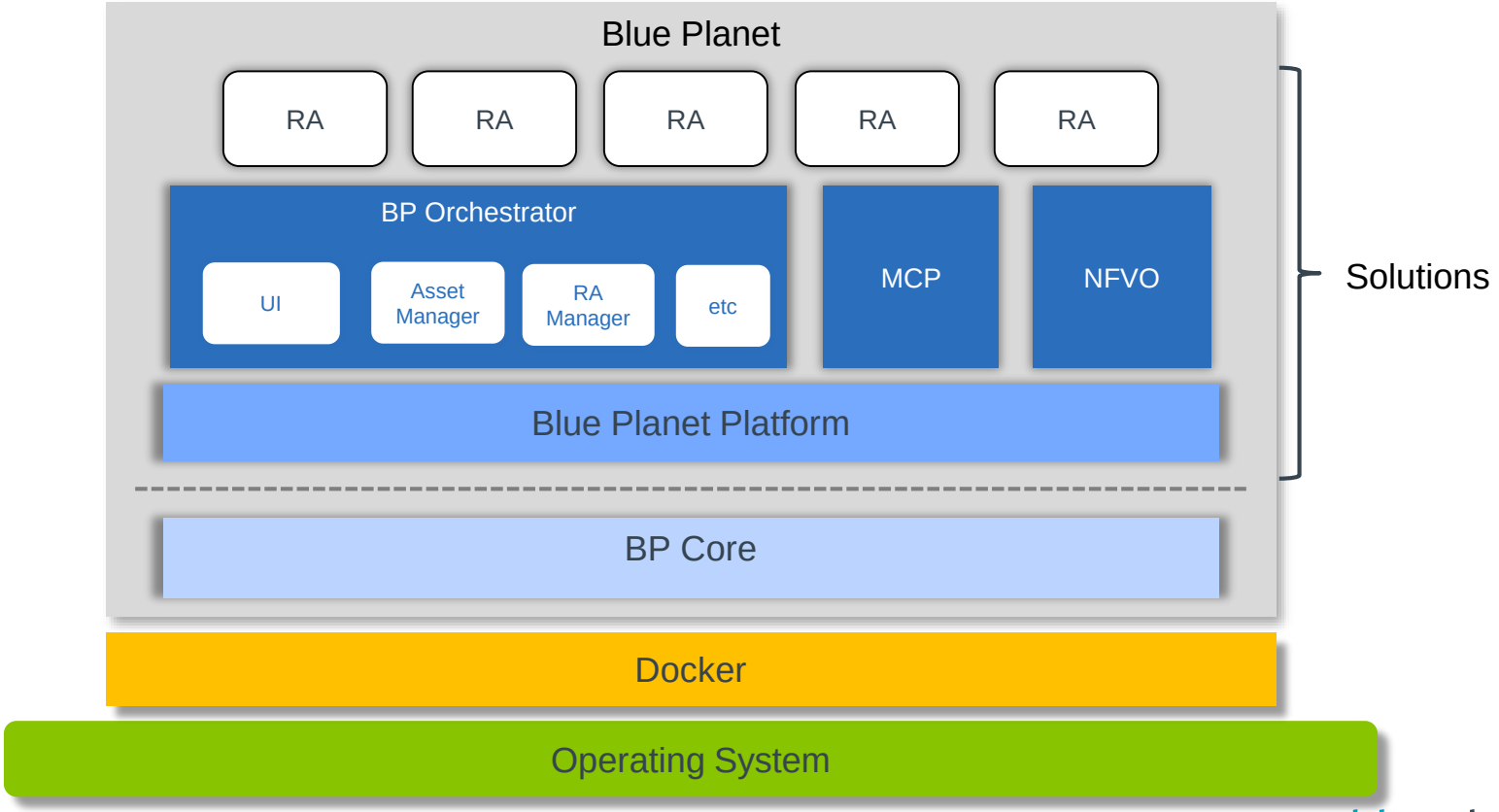


blueplanet

Blue Planet Infrastructure

- Core Platform Components:
  - Docker
  - Solution Manager
    - Docker container to manage component deployment
    - Components are called ‘Solutions’
  - Blue Planet Platform solution
    - Facilitates messaging and communication between other Blue Planet components
  - Micro-Services and Resource Adapters run as “solutions”

Blue Planet Architecture



blueplanet

Blue Planet Resource Adapter Architecture

- Philosophy
  - Micro-services and Container based
  - Leverage the open source community
- Ability to scale
  - Number of elements managed by RA
  - Scale of RA development itself
- RAs are agnostic of consuming applications
  - Can be called when needed or publish autonomously to message bus
- Don’t Block Progress, be Extensible
  - Make sure developers are not blocked by lack of support in RA libraries
  - Framework supports extensibility through call outs if functionality is missing

Blue Planet Resource Adapter Architecture

- Micro-Service Based Architecture
  - REST Interface for Synchronous Communication
  - Message bus for Asynchronous Communication
- RACTRL
  - Separate micro-service for aggregation and discovery of RAs
  - Manages connections between Blue Planet and devices/domains

Blue Planet Developers Exchange

- Blue Planet Developer Community
  - community.ciena.com
  - Facilitates collaboration between customers and ecosystem partners using the DevOps Toolkit
- Blue Planet RAs and Service Templates can be published and shared within the Community
  - Open source-like model; “Libraries" of resources
  - Programming tools, training and documentation
- Blue Orbit partners and other solution providers can access the Toolkit and the Community
  - Partners develop and update their own RAs
  - Encourages open collaboration
  - Ensures production-grade interoperability

blueplanet

Blue Planet DevOps Toolkit

- DevOps Toolkit is provided for training as part of class.
- Students get a three month license to use the DevOps Toolkit for training purposes.
  - Please contact your sales team if you require access to other versions of the DevOps Toolkit, require to use the Devops Toolkit for more than three months or require access to areas of the DevOps Exchange that are restricted to those with full license access.

What's In the DevOps Toolkit?

- Vagrant box with all the software necessary to develop RAs
  - Ubuntu VM
  - rasdk-python: Ciena's Resource Adapter Software Development Kit for the Python language
  - Docker images:
    - **bpocore-dev**: Development version of Blue Planet Orchestrator
    - **orchestrate-ui-dev**: Web UI for testing Service Templates
    - **script-dev**: For creating remote scripts (related to service templates)
  - Git
  - Docker
- Requires Vagrant, Virtualbox and a ssh client on your host computer

Docker essentials

- Docker: Like a VM, but lightweight
- Essential commands:
  - **`docker run`** - creates a container and then runs it
  - **`docker start`** - starts an existing container
  - **`docker stop`** - stops an existing container
  - **`docker ps`** - shows container that are running (use -a to show all, including not running)
  - **`docker logs`** - view logs
  - **`docker rm`** - deletes a container

BP docker scripts

- **`docker-compose up`**:
  - **`docker run`** for bpocore-dev, frost-orchestrate-ui-dev and script-dev
  - Runs bpocore-dev in foreground
  - Can remove all with Control-C
  - Based on **docker-compose.yml** file
- **`docker-compose kill`**:
  - **`docker stop`** for bpocore-dev, frost-orchestrate-ui-dev and script-dev
- **`docker-compose rm`**:
  - **`docker rm`** for bpocore-dev, frost-orchestrate-ui-dev and script-dev
  - Containers must first be stopped

BP docker scripts

- Prior to 17.06:
  - **`blueplanet-start`**: **`docker run`** for bpocore-dev & frost-orchestrate-ui-dev
  - **`blueplanet-stop`**: **`docker rm -f`** for bpocore-dev & frost-orchestrate-ui-dev
  - Note: no script-dev in previous versions
  - These scripts still work in 17.06

blueplanet

Lab: Setup the Blue Planet DevOps Toolkit

- Complete the following labs:
  - Optional lab 1: Setup the Blue Planet DevOps Toolkit
  - Lab 2: Starting the Orchestrate Docker Containers
  - Lab 3: Test RA Onboarding
  - Lab 4: Suspending and Restarting the Development Environment

Notify your instructor when you have completed these labs.

blueplanet

blueplanet
a division of ciena

Resource Adapters

Resource Adapters - Preview

- Topics covered in this section include:
  - Introduction to bpprov and RASDK
  - The steps to creating a Resource Adapter
  - Overview of the rasdk-python and rpsdk-python packages
  - Development tools

blueplanet

RA Created with RASDK Overview

- The heart of the RA is the **bpprov** library
  - Manages southbound connections to devices or domain controllers (e.g. CLI, NetConf, SNMP).
  - Executes a command pipeline to push/pull data to/from the southbound network resources.
- RASDK wraps bpprov with a REST API and general session management.
  - Provides a northbound REST API
  - Southbound connects-to, monitors, and translates data from southbound network resources.

RASDK Overview

- Goal:
  - Allow developers to focus on bpprov command implementations
  - Provide common tools for APIs, session management, data transformations, and testing
  - Minimize initial configuration

blueplanet

Creating RAs with the DevOps Toolkit

1. Render a new RA project
2. Customize your RA
3. Add a Resource Provider to your RA
4. Run locally, test, and iterate against bpocore-dev
5. Make and deploy the solution to your Blue Planet server

blueplanet

Creating RAs with the DevOps Toolkit

- **ra**sdk-python
  - Collection of Python libraries which can be used to implement a Blue Planet resource adapter
  - Looks like a typical python project and uses well known python tools
  - Distributed using **setuptools**
  - Leverages standard python tools **pip** and **virtualenv**
    - virtualenv isolates your application's python packages from already installed python packages
    - setuptools and pip are used to install the python packages
- **rp**sdk-python
  - Tools for creating a Resource Provider
  - Python (Twisted) library for building Resource Providers (RPs) to interact with Blue Planet Orchestrate Core (bpocore).

virtualenv

- virtualenv – tool to isolate Python environments
- Allows you to have multiple Python project, each with separate required packages
  - Project 1 can require LibFoo version 1
  - Project 2 can require LibFoo version 2
  - virtualenv keeps them isolated
- Helps keep your global site-packages directory clean and manageable
- https://virtualenv.pypa.io/en/stable/

Suggested Development Tools

- Text editor
- Python IDE
- JSON Validator
  - www.jsonlint.org or other
  - env/bin/bpprov-cli validate-json *dir*
- SSH
  - SSH is required to access the Vagrant environment
- cURL, Postman or similar REST client
  - Not required, but useful for testing your RA

Blue Planet Resource Adapter (RA) Development

Agenda

1. Blue Planet and RA Development Overview
2. Creating an RA Project
3. bpprov Overview
4. Creating a CLI-Based RA
5. Customizing an RA

blueplanet

blueplanet
a division of ciena

Creating an RA Project

Creating an RA Project - Preview

- Topics covered in this section include:
  - The **paster** utility
  - The **settings.py** file
  - The **make** command
  - Running a RA in the Devops TK
  - Communicating with a RA via REST APIs

blueplanet

Render a New RA with rasdk-paste

- rasdk-paste package provides:
  - Python utility for generating a new RA project from a template
  - Based on PasteScript Python Project
  - Creates the RA project in the current folder:

```
~/shared$ paster
```

- **Note**: Do not run in an existing RA project folder

blueplanet

Important Paster prompts

```
vagrant@vagrant:~/shared$ paster
The name of the RA [MyRA]: RA-docker 1
Python package name for your RA [radocker]: 2
The author of the RA [Ciena Engineer]: Bo Rothwell
The vendor of the RA [Ciena]: Ciena
List the endpoints supported by your RA. [['cli', 'snmp']]: ['cli'] 3
General type or class of things the RA connects to, e.g. Ciena_Optical. [Demo]: 4
List the subgroup or family of devices/domains that your RA manages, e.g. Ciena6500 [['Demo-A' 5
]]:
List the specific types of devices/domains supported by this RA, e.g. CN6500 [['DEMO-A-1']]: 6
Select your RA's configuration optimization settings:
1 - no - don't optimize
2 - yes - use YANG model-driven configuration optimization
Choose from 1, 2 [1]:
Select if you'd like the virtualenv directory initialized for you:
1 - no
2 - yes
Choose from 1, 2 [1]: 7
```

1. Top level project directory and the overall RA name
2. Configuration directory and name of RA script
3. ENDPOINTS – Should be a list value
4. TYPE_GROUP - the general type of resource
5. RESOURCE_TYPES - the specific types of resources
6. Device type– Specific type of device
7. Install the RA Python software or no?

blueplanet

Important Paster prompts

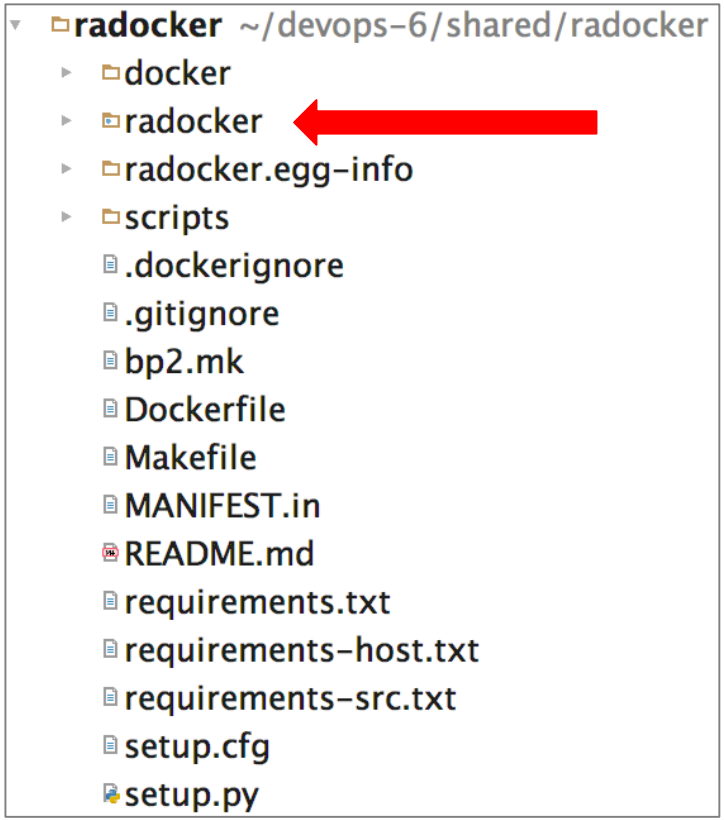
```
Will you be running the RA as an Orchestrate resource provider? [Y/n]: 1
Organization which maintains the RA. (URL is recommended) [http://ciena.com]:
The vendor's license for the RA. (URL is recommended). [http://ciena.com/license]:
RP ID (must be a valid UUID or urn such as urn:ciena:bp:ra:ciena6500) [urn:ciena:bp:ra:radocke
r]:
ID for the RP's first domain (must be a valid UUID or urn such as urn:ciena:bp:domain:ciena650
0) [urn:ciena:bp:domain:radocker]:
Maintainer/owner of this RP [Bo Rothwell]:
Email for the maintainer [bo.rothwell@email.com]: brothwel@ciena.com
Human-readable title of this RP [RA-docker]:
Describe the RP []:
RP version [0.0.1]:
Select your resource provider type. Please consult the rpsdk section of https://git.blueplanet
.com/DevOpsToolkit/dtkdist/tree/master/docs for more information:
1 - domain manager
2 - device manager
Choose from 1, 2 [1]: 2
```

1. Configure Resource Provider now or no?
2. Does RA talk to devices or other Orch products?

blueplanet

What paster creates for a new RA

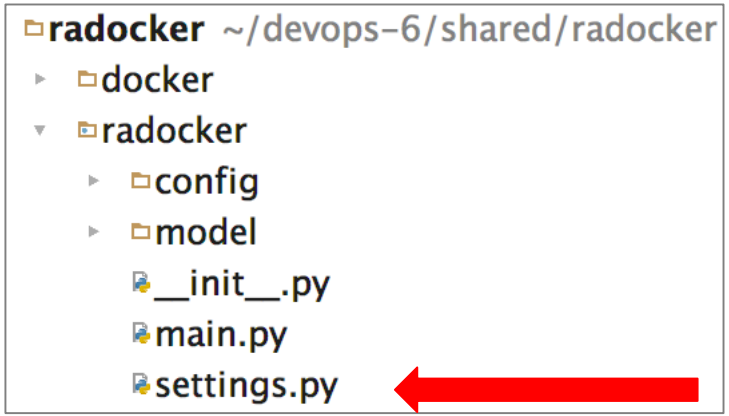
- RA template can be run on the BP Platform (as a docker image)
  - The RA package is a standard python package that uses the setup.py file
  - A **Makefile** is provided for typical actions: prepare-venv, clean, etc.
  - The RA package code is in directory using package name
    - `radocker/radocker` in this example



blueplanet

settings.py

- Key values designed for basic RA and RP configuration
- View ***RA*/env/lib/python2.7/site-packages/rasdk/default_settings.py** for all options
  - This file only exists after running the **`make prepare-venv`** command
  - Most settings you need are already set in **settings.py**



blueplanet

Key RA values of settings.py file

- DEFAULT_PORT:
  - Port number RA script listens to when running in Devops TK
  - To test multiple RAs, be sure to use different ports
- RA_NAME: The name of the RA
- TYPE_GROUP: The general type of resource supported by the RA
- RESOURCE_TYPES: lists the specific types of resources supported by the RA
- ENDPOINTS: The communication protocols supported by the RA

Prepare the New RA

- Simple **Makefile** for the virtual environment
  - Includes several **make** targets
  - Example:

```
$ make prepare-venv
```

- `make prepare-venv`
  - **setup.py** lists packages
  - **requirements.txt** captures required libraries and versions
    - Can point to git repos or pypi server
    - All python libraries will be installed (Internet connection is required)

blueplanet

Understanding make

- The `make` utility allows for management functions, such as compiling code or installing software
- The `make` utility in the Devops TK is used to manage Python software
- Uses the **Makefile** file which is located in the root of the project directory
- The **Makefile** contains *targets*, which are used to specify what actions to take
- Common `make` targets:
  - `make all` – List all make functions (note: not all enabled in the Devops TK)
  - `make clean` – Remove previously installed Python software

RASDK Python Package

- Contents of RASDK includes:
  - Most of the application code needed to run an RA
  - Cyclone web server
  - Access to Kafka message bus
  - cymlrest model driven REST interface
    - CyML: Cyan Modeling Language
  - SNMP trap client support
  - bp-prov – a template driven framework for RAs

blueplanet

Run Your RA

- Execute RA with **`env/bin/`*`raname`*** from top level project directory
- Example:

```
radocker$ env/bin/radocker
```

- Interact with RA:
  - radocker will be running on a web server on port 8080

```
curl -s http://localhost:8080/api/v1/raInfo | python -m json.tool
```

  - Swagger: http://192.168.33.10:8080/api/v1/swagger-ui/

blueplanet

REST API Resources

- raInfo:
  - Basic info about the RA
- resourceTypes:
  - Overall schema, Network Element = ‘device’
  - Specific types the RA talks to, Ex: ARS9000, ARS903, etc.
- typeGroups:
  - General types the RA can talk to, Ex: Cisco
- Sessions
- JSON Schemas
  - connectionSchema
  - authentication Schema

blueplanet

Test the RA via the Swagger UI

- Select raInfo -> Get /raInfo

blueplanet

Lab: Create an RA Project

- Complete the following lab:
  - Lab 5: Create an RA Project

Notify your instructor when you have completed these labs.

blueplanet

blueplanet
a division of ciena

Customizing an RA Project

Customizing an RA Project - Preview

- Topics covered in this section include:
  - Overview of the process of customizing an RA
  - Review of **`git`**
  - TYPE_GROUP, RESOURCE_TYPES, DEVICE_TYPES and ENDPOINTS
  - The **family.json** and **device.json** files
  - The **endpoint-params.json** and **endpoints.json** files
  - A review of Regular Expressions
  - An overview of command files
  - The **`bpprov-cli`** command

blueplanet

Customizing Your RA

- Create a Git repo (optional)
- Configure **settings.py**
  - TYPE_GROUP
  - RESOURCE_TYPES
  - ENDPOINTS
- Configure bpprov
  - Define family and devices
  - Customize endpoints
- Create commands

blueplanet

Snapshot the Default RA

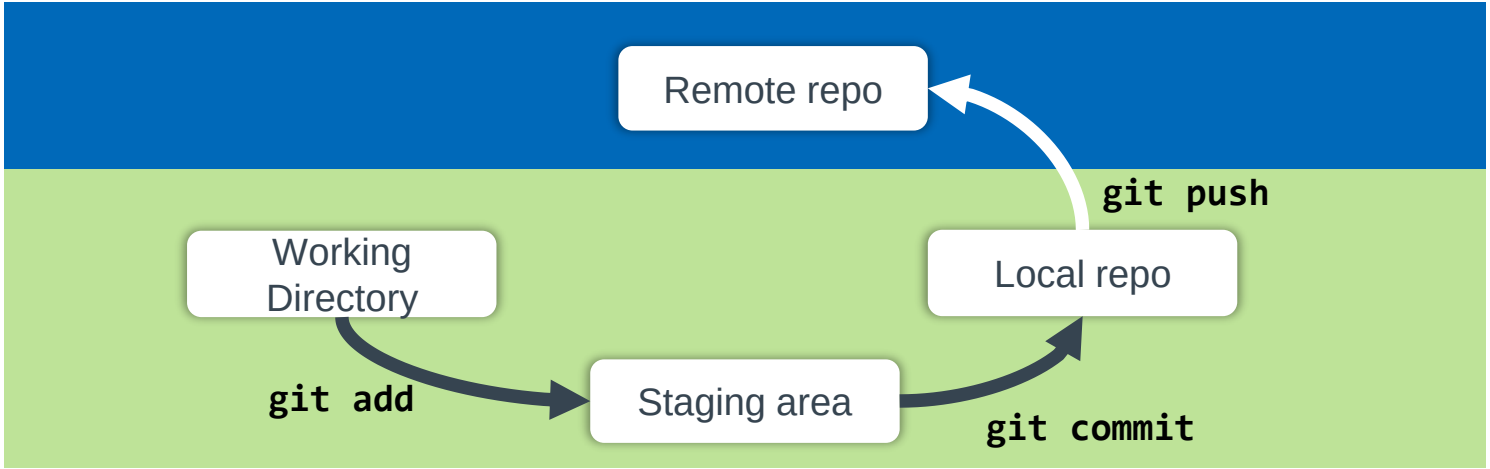
- rasdk is setup to be managed in **`git`**
  - A good starting **.gitignore** file is included in your generated RA project.
- Use **`git`** to save a baseline of the newly created RA
- Creates a git repository in subdirectory ***RA*/.git**

```
$ git config --global user.email "training@ciena.com"
$ git config --global user.name "Joe Engineer”
$ git init
$ git add –A
$ git commit -m "initial docker RA"
```


blueplanet

Understanding git

- Version control software
- **`git init`** - initializes a new project in current directory (called the “working directory”)
- **`git add`** - adds changes to files in working directory to the “staging area”
- **`git commit`** - places changes from working directory to local repository
- **`git push`** - places changes from local repository to remote repository



Remote repo (BPO Docker Container)

Virtual OS (Local VM)

blueplanet

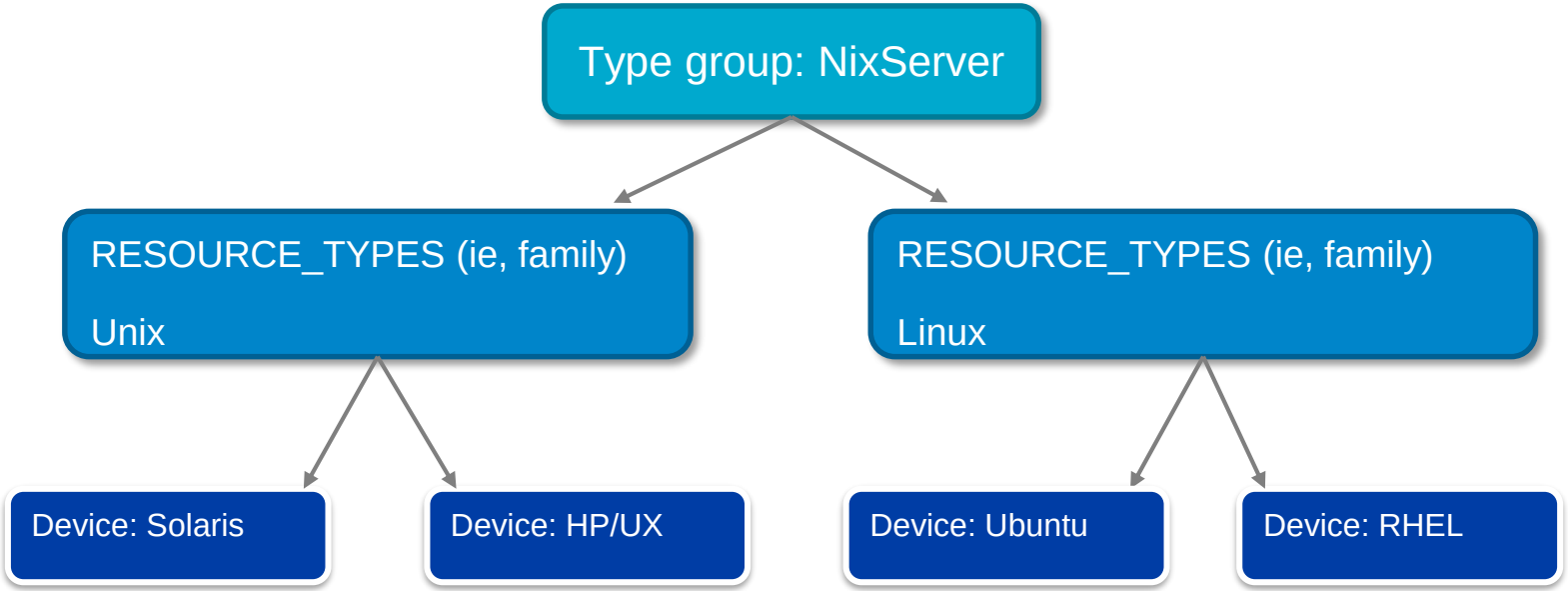
Additional git commands

- **`git clone`** - initializes a new project based on files from remote repository
- Remove file from working directory and local repository:
  - **`git rm file1.txt`**
  - **`git commit -m "remove file.txt"`**
- Remove file only from local repository:
  - **`git rm --cached file.txt`**

blueplanet

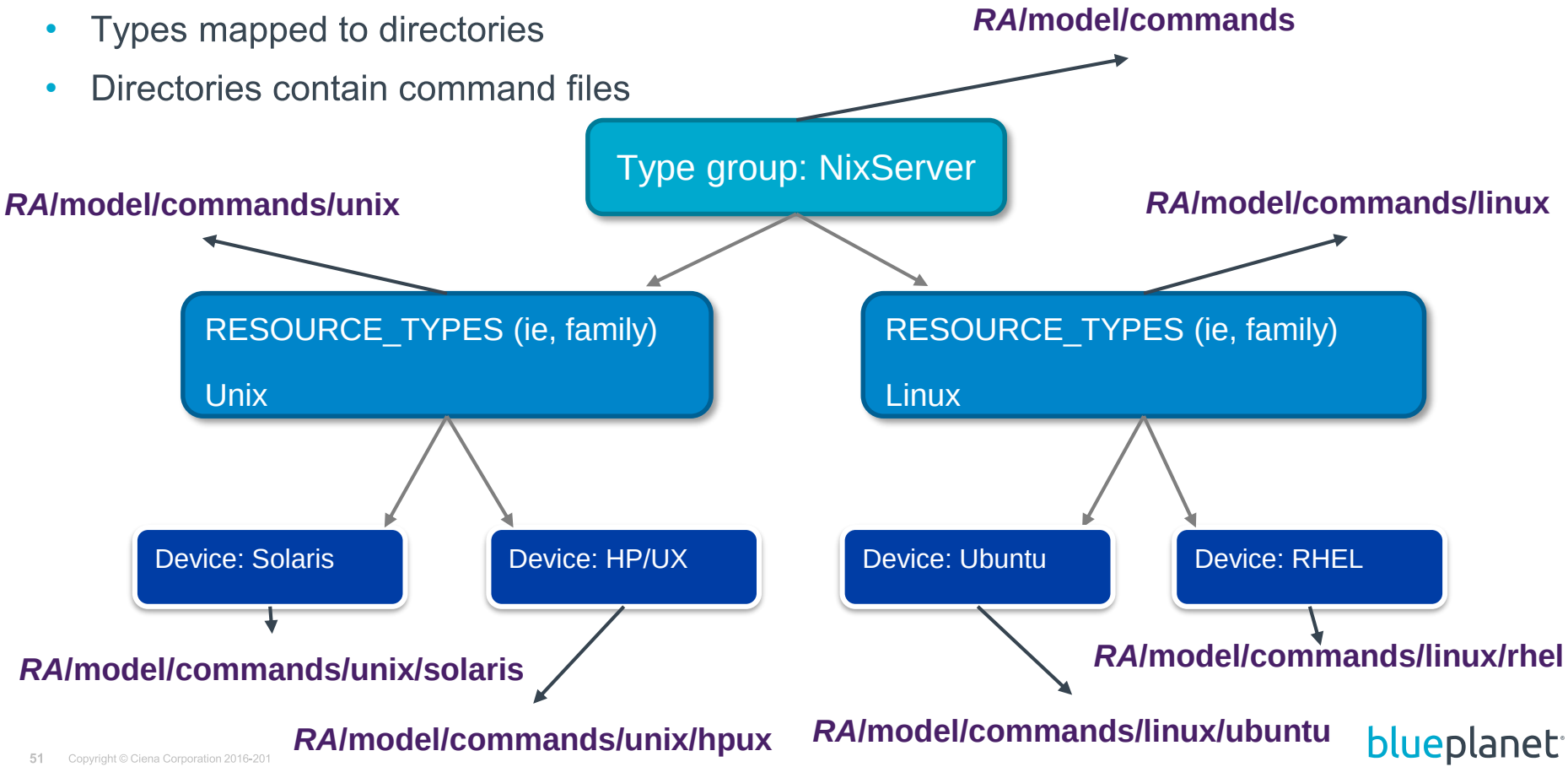
Defining Types

- RESOURCE_TYPES = AKA family
- DEVICE_TYPES = AKA device



Mapping

- Types mapped to directories
- Directories contain command files



blueplanet

Configure settings.py

- See ***RA*/env/lib/python2.7/site-packages/rasdk/default_settings.py** for all available settings
- TYPE_GROUP and RESOURCE_TYPES ***must*** be globally unique
- **TYPE_GROUP** – Generic family of devices, e.g. 'NixServer'
  - TYPE_GROUP is used by RACTRL to assign a session to the RA.
  - A TYPE_GROUP can only exist in a single RA so that RACTRL can assign the session to the correct RA
  - Recommend using Python Identifier compliance for the TYPE_GROUP value
    - https://docs.python.org/2/reference/lexical_analysis.html#identifiers

Configure settings.py

- RESOURCE_TYPES – Specific type of device within the family, e.g. 'NixDockerServer'
- RESOURCE_TYPES must also match entries found in the bpprov related settings files:
  - **device.json**
  - **family.json**
- TYPE_GROUP is a string because only one group per RA
- RESOURCE_TYPES is a list because there can be multiple types per group

```
TYPE_GROUP = 'NixServer'
RESOURCE_TYPES = ['NixDockerServer', ]
```

Configure settings.py (cont)

- ENDPOINTS
  - rasdk default is cli and snmp management
  - Current valid ENDPOINTS: 'AMQP', 'cli', 'snmp', 'tl1', 'netconf', 'soap', 'rest', 'xml_rest', 'null', 'web socket', 'NetconfYang‘, and 'kafka’
  - Additional ENDPOINTS may be added in future releases
  - snmp and cli are the defaults:

```
ENDPOINTS = [
    'cli',
    'snmp',
    ]
```

Customizing bpprov

- bpprov is the template-based data transformation library.
- It is part of the RA
- All bpprov settings and command pipelines are located in the **model** directory.
- The majority of changes to the RA are made within this subdirectory.
- Start by:
  - Verify the RESOURCE_TYPE in the family.json file (***RA*/*raconf*/model/family.json**)
  - Verify the DEVICE_TYPE to the device.json file (***RA*/*raconf*/model/device.json**)

blueplanet

Customizing bpprov

- The DEVICE_TYPES and RESOURCE_TYPES drive the directory hierarchy in the various model subdirectories.
  - Assign the current directory (".") to *dir* in both files to keep the directory structure flat and simple.
- The **family.json** and **device.json** file control the bpprov commands lookup hierarchy.
  - Best Practice: Organize commands by RESOURCE_TYPES and DEVICE_TYPES

family.json

Must match RESOURCE_TYPES in **settings.py**

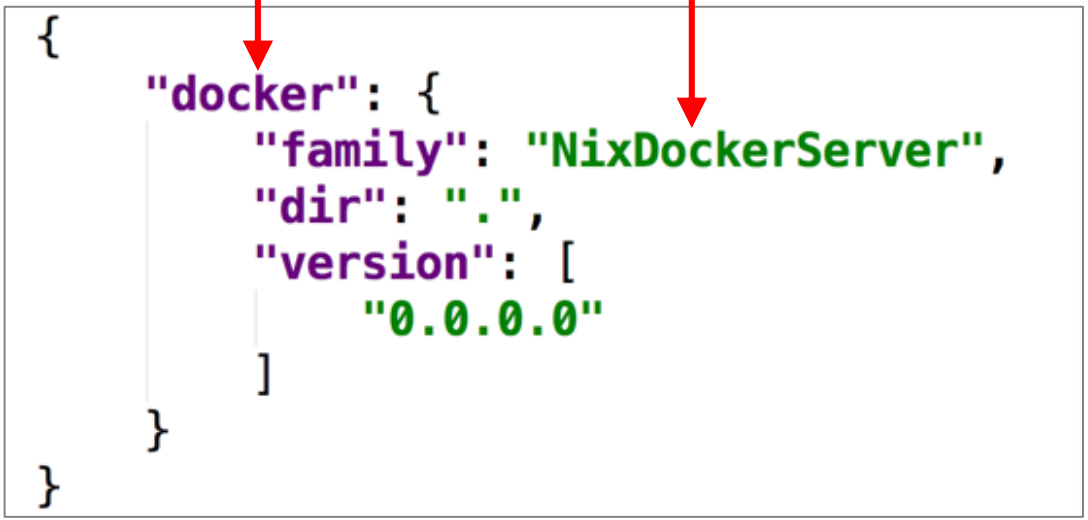
```
{
    "NixDockerServer": {
        "dir": ".",
        "displayName": "NixDockerServer"
    }
}
```

- **dir** value is relative to ***RA/raconf*/model/commands** directory
- Can include multiple values (RESOURCE_TYPES)

device.json

DEVICE_TYPE

Must match RESOURCE_TYPES in **settings.py**



```
{
    "docker": {
        "family": "NixDockerServer",
        "dir": ".",
        "version": [
            "0.0.0.0"
        ]
    }
}
```

- **dir** value is relative to ***RA/raconf*/model/commands** directory
- Can include multiple values (DEVICE_TYPEs)

blueplanet

Device Versions

- Version can be:
  - An array of strings
  - An object specifying version and directory for commands
    - Allows developer to provide different commands for different versions
    - Version dir is relative to DEVICE_TYPE dir

```
{
 “docker": {
   "family": "NixDockerServer",
   "dir": ".",
   "version": {
      "1.2.3": { "dir": "1_2_3" },
      "1.2.5": { "dir": "1_2_5" }
   }
  }
}
```


blueplanet

Setting RESOURCE_TYPES in bpprov

*RA/raconf/*settings.py

```
TYPE_GROUP = 'NixServer'
RESOURCE_TYPES = ['NixDockerServer', ]
```

*RA/raconf/*model/family.json

```
{
 "NixDockerServer": {
    "dir": ".",
    "displayName": "NixDockerServer"
  }
}
```

*RA/raconf/*model/device.json

```
{
 "docker" {
   "family": "NixDockerServer",
   "dir": ".",
   "version": [
       "4.5.6.7s"
    ]
  }
}
```


blueplanet

Customize Endpoints

- An endpoint: component that specifies the communication protocol between the RA and the device
- RASDK communicates with endpoints using session parameters and fixed parameters that are set in the ***RA*/*raconf*/model/endpoint-params.json** file
  - Default connection for CLI is ssh
  - Can be set to telnet
- Update ***RA*/*raconf*/model/endpoint-params.json** with correct settings
  - Settings must match device
  - Device CLI prompt must match regular expression (regex)

Sample – cli endpoint

```
{
    "cli": {
        "commandTimeout": 30,
        "heartbeatInterval": 30,
        "reconnectPeriod": 30,
        "connectTimeout": 30,
        "prompt": "^[A-Za-z0-9\\-@()\\~:]+[>#\\$]",
        "postLoginCommands": [
        ],
    }
}
```

- How long (in sec) to wait on each command before timing out
- How often (in sec) to send periodic heartbeat message
- When disconnected, how long (in sec) to wait before trying to reconnect
- How long (in sec) to wait on initial connect before timing out
- Regex that identifies the prompt pattern
- Set of additional commands to execute post authentication phase

blueplanet

Sample – rest endpoint

- Note: refer to bpprov documentation (BP-Prov Developers Guide) for details regarding these settings

```
{
    "rest": {
        "hostname": "localhost",
        "hostport": 5000,
        "scheme": "http",
        "auth": {
            "type": "bpprov.components.rest_auth.Null",
            "parameters": {
            }
        }
    }
}
```

Regular expressions

- Critical to understand for RA development - used in several configuration files
- Designed to match text using symbols
- Resource: http://www.rexegg.com/regex-quickstart.html/
- Tester: http://regexr.com/
- Example: `^[A-Za-z0-9\\-@()\\~:]+[>#\\$]`
  - Note: \\ = a single \ in JSON files

blueplanet

Important Regular Expressions

RE	Description
^	Match the beginning of text
$	Match the end of text
+	Repeat the previous one or more times
*	Repeat the previous zero or more times
\	Escape special meaning of next character
[ ]	Match one character from set within [ ]

RE	Description
\d	Match a single digit character
\D	Match a single non-digit character
\s	Match a single white space character
\S	Match a single non-white space character
.	Match a single non-newline character
?	Match an optional character

- Examples:
  - **^\d+$** - match text that only contains numbers
  - **^[a-z]\$.*xyz$** - match text that starts with a lower case letter, followed by a “$”, followed by zero or more of any characters, followed by “xyz” and the string ends.

Creating Commands

- Commands are specified in JSON files
  - Filename: **command-name.json**
- Commands go in ***RA*/*raconf*/model/commands**
  - Commands should be separated by family (RESOURCE_TYPE) and device (DEVICE_TYPE) into sub-directories
  - Commands that are common to multiple devices go in the ***RA*/*raconf*/model/commands** directory.

Debuggers: bpprov-cli

- Tool for testing command files
- Creates a temporary session using ***RA/raconf*/model/endpoints.json**
- endpoints.json:
  - Only used with **`bpprov-cli`** for RA testing
  - Contains session parameters – should match **endpoint-params.json**

```
env/bin/bpprov-cli command-run RA/raconf/model cli
commands/show-docker-version.json '{}' --start
```


blueplanet

Lab: Customize an RA Project

- Complete the following lab:
  - Lab 6: Customizing Your RA

Notify your instructor when you have completed these labs.

blueplanet

Blue Planet Resource Adapter (RA) Development

Agenda

1. Blue Planet and RA Development Overview
2. Creating an RA Project
3. bpprov Overview
4. Creating a CLI-Based RA
5. Customizing an RA

blueplanet

blueplanet
a division of ciena

bpprov Fundamentals

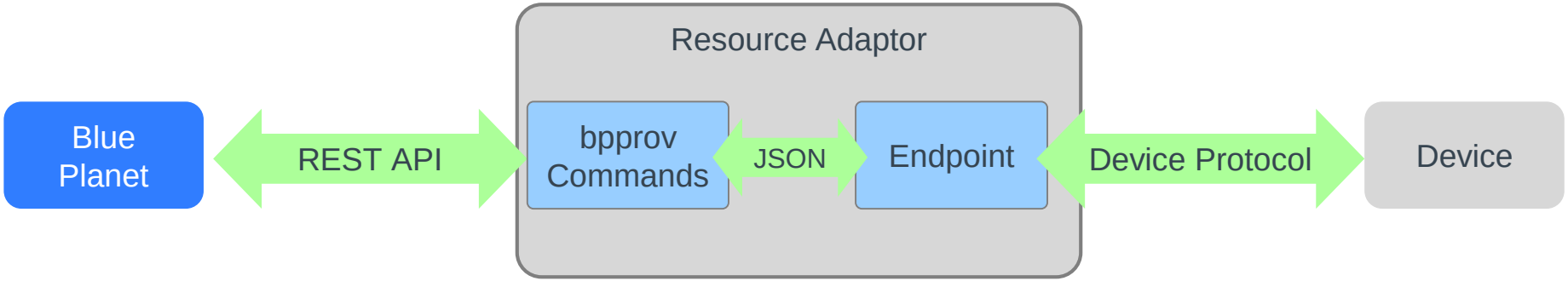
bpprov Fundamentals - Preview

- Topics covered in this section include:
  - Data flow from BPO to device
  - JSON Schemas
  - Introduction to Runners and Translators
  - Overview of Route Data
  - Endpoint Consumer and Producer

blueplanet

Role of Resource Adapters

- Translates requests from Blue Planet into an object model and protocol that the domain understands



Introduction to bpprov

- To develop a Resource Adapter (RA), developers need to deal with a set of common tasks:
  - Identify the communication protocol to talk to the device (CLI, REST, etc.).
  - Identify the set of commands that need to be sent to the device in order to perform CRUDL operations.
  - Understand the CRUDL input/output, and in some cases also learn how to properly parse them.
  - Finding out how to properly transform the data from the device to a common structure that the application expects.
- bpprov captures these commonalities, and creates reusable components across all RAs

Customizing bpprov

- bpprov is the template-based data transformation library
- All bpprov settings and command pipelines are located in the ***RA/raconf/*model** directory.
  - The majority of changes to the RA are made within this subdirectory.
- JSON is the standard within bpprov
  - All structures are JSON-derived structures
  - bpprov commands are JSON files
  - JSON Schema used to enforce JSON formats

JSON schema

- Describes the valid format of specific JSON files
- Critical to understand when building RAs
- Resource: http://json-schema.org/

blueplanet

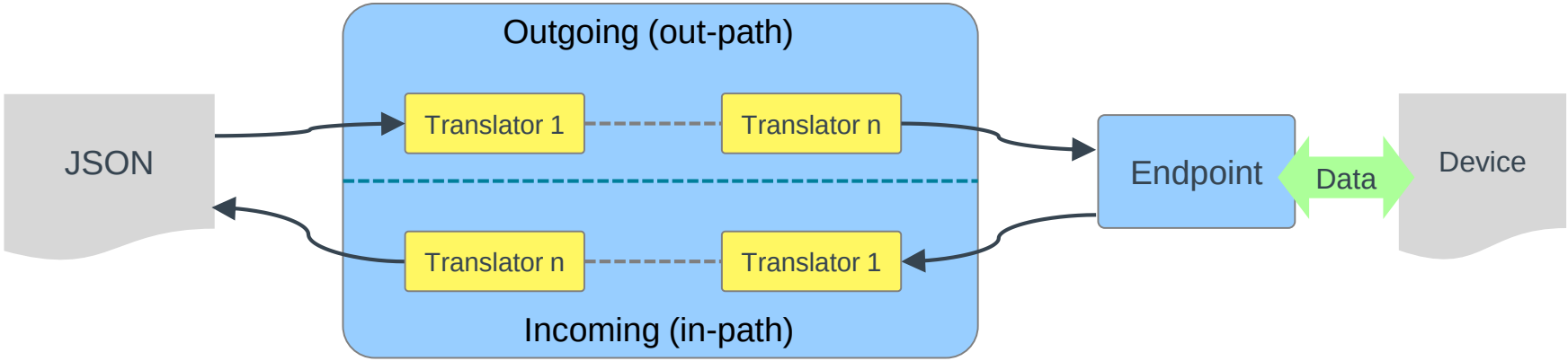
JSON schema example

```
{
        "title": "Example Schema",
        "type": "object",
        "properties": {
                "firstName": {
                        "type": "string"
                },
                "age": {
                        "description": "Age in years",
                        "type": "integer",
                        "minimum": 0
                }
        },
        "required": ["firstName"]
}
```


blueplanet

Runners

- bp-prov commands contain two paths: *in* and *out*
- Each path has a runner
  - Runner is set of translators that manipulate data



blueplanet

Classifying Translators

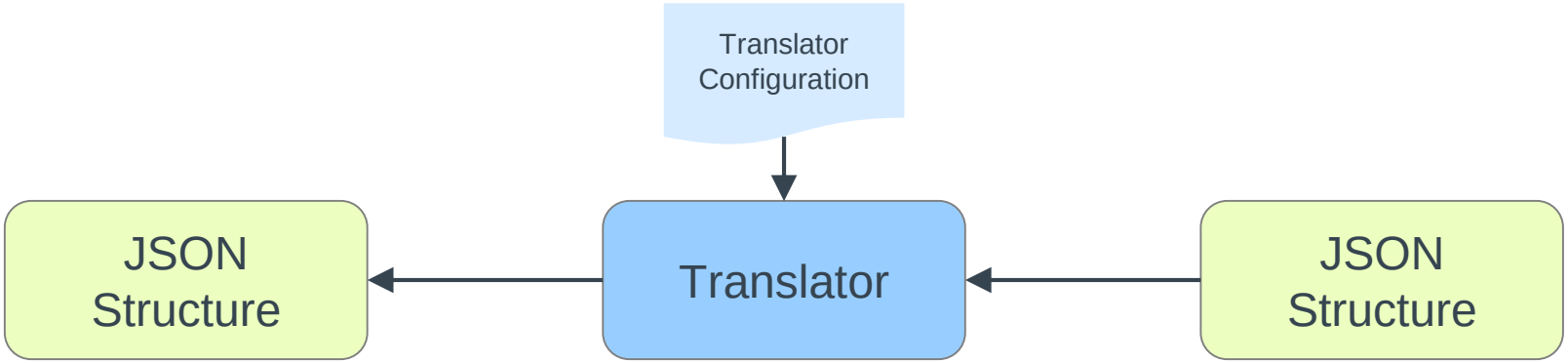
Transform	• Identify certain data inside the JSON structure, and replace it with another value.
Filter	• Identify certain pattern within the data and make decisions whether the data should be sent to the next pipeline.
Aggregate	• Call external commands and, based on the result of the command, pick and choose which data to aggregate.
Branch	• Decide to run certain translation, or skip it.
Template Construct	• In the case of a complex structure, e.g. hierarchical tree, translator can use template engine to help construct a more arbitrary structure.

bpprov Translators

- bp-prov comes with a rich set of translators for common use cases
- Each translator has the following attributes:
  - Schema: Defines how the translator parameters are structured. The schema is also used internally by the translator to validate its parameters.
  - Parameters: Defines the behavior of the translator. Most translators relies on the parameters attribute to allow the user to change the translator's behavior.
- See Blue Planet Developer Guide for a list of all translators and how to use them.

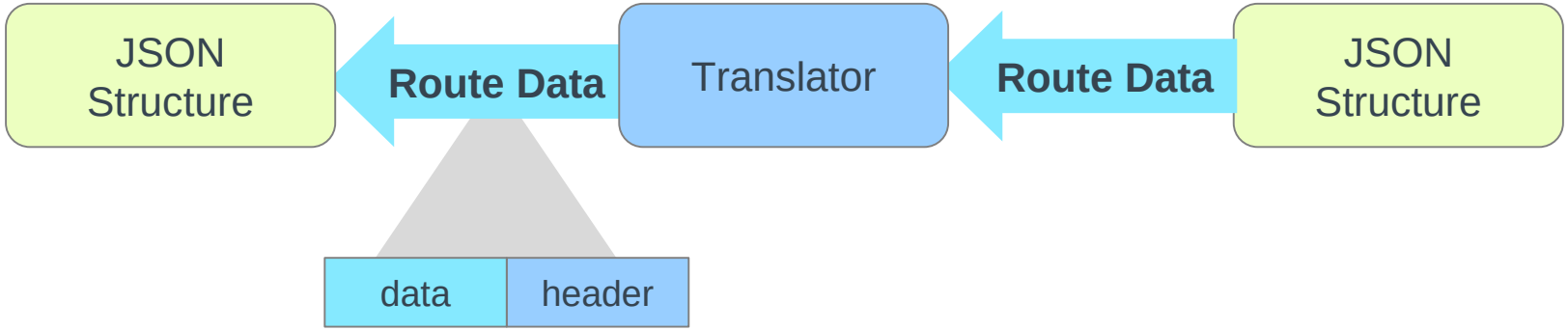
Translators

- Transform data from one form to another
  - Data manipulation utilities within bpprov



Route Data

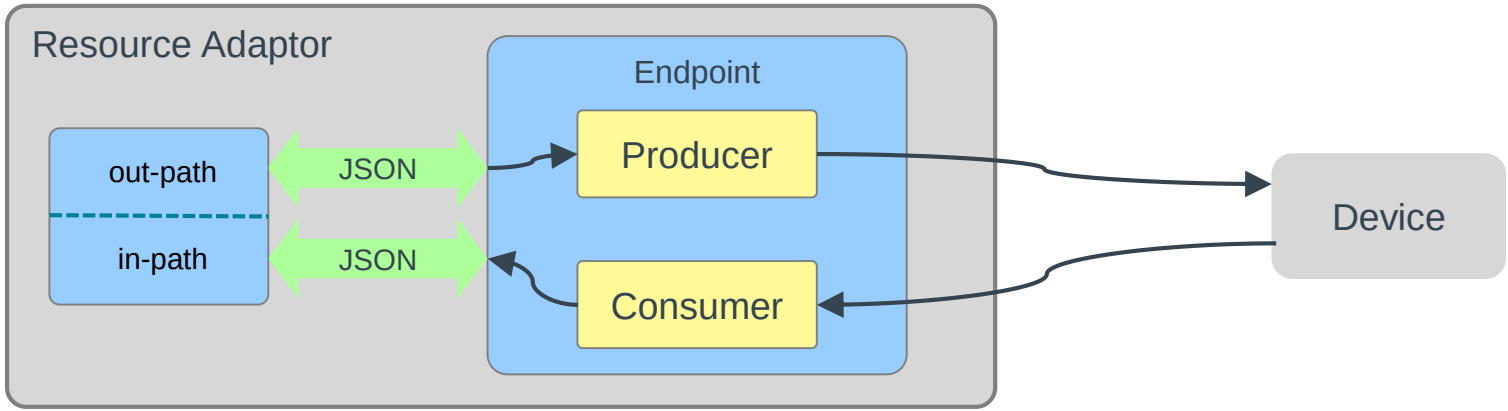
- The internal data structure that is being transformed by a translator



  - Data: JSON structure being translated
  - Header: Used internally to pass messages

Endpoint

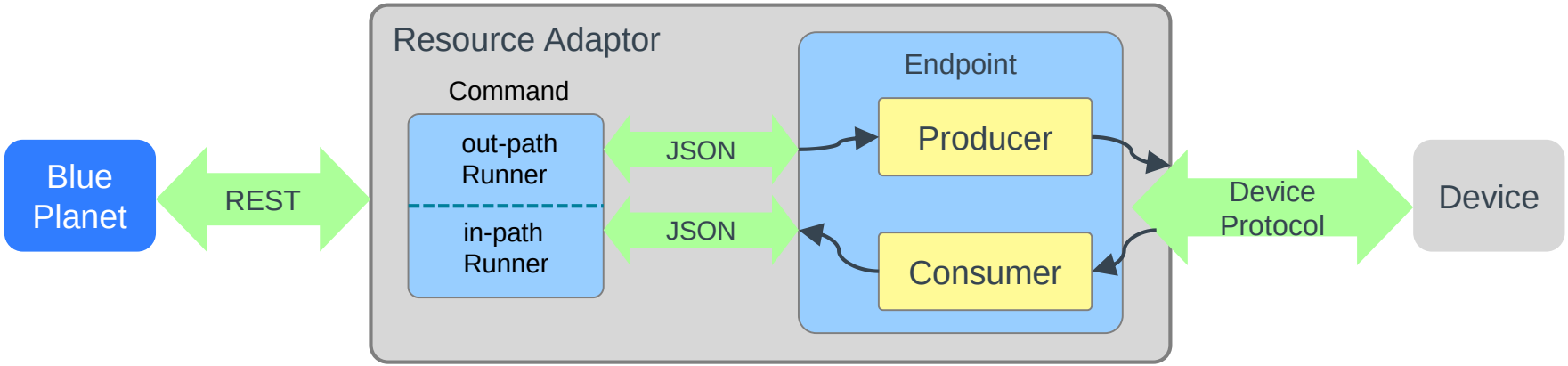
- Component that specifies the communication protocol between the RA and the device/domain
- Producer: Translates JSON structure to the device
- Consumer: Parse data from device into a JSON structure



blueplanet

Building Commands

- Commands are defined in JSON files
- Most work is building Runners
- Commands go in the ***RA/raconf*/model/commands** directory



Blue Planet Resource Adapter (RA) Development

Agenda

1. Blue Planet and RA Development Overview
2. Creating an RA Project
3. bpprov Overview
4. Creating a CLI-Based RA
5. Customizing an RA

blueplanet

blueplanet
a division of ciena

Creating a CLI-Based RA

Creating a CLI-Based RA - Preview

- Topics covered in this section include:
  - Review the process of creating an RA
  - Next lab will be to create a new RA from scratch which will be used for the remaining class labs

Creating RAs with the DevOps Toolkit

1. • Render a new RA project
2. • Customize your RA
3. • Add a Resource Provider to your RA
4. • Run locally, test, and iterate against bpocore-dev
5. • Make and deploy the RA docker image to your Blue Planet server

blueplanet

Initial Customization

- Render a new RA using paster
- Configure **settings.py**:
  - TYPE_GROUP
  - RESOURCE_TYPES
  - ENDPOINTS
- Update **family.json**
- Update **device.json**
- Update **endpoint-params.json**
- Create a git repo

Plan Your RA

- The name of the RA
  - ra-<some name>
  - paster creates a folder using the project name, e.g. *ra-ciena6500*
  - paster removes the hyphen for the RA package, e.g. *raciena6500*
- The TYPE_GROUP
  - Only one is needed for class lab: *DockerContainer*
- The RESOURCE_TYPES
  - Only one is needed for class lab: *docker*
- The DEVICE_TYPE
  - Only one is needed for class lab: *linux*
- Where commands will go
  - ***RA/raconf*/model/commands/<RESOURCE_TYPE>/<DEVICE_TYPE>**

blueplanet

rasdk-paste Command

- Run **`paster`**
- Enter RA information into prompts:

```
~/shared$ paster
```


blueplanet

Prepare and Test Your RA

- Run the **make** command:

```
<your-ra>$ make prepare-venv
```

- Run your RA:

```
<your-ra>$ env/bin/raname
```

- Test your RA:

```
$ curl -s http://localhost:8080/api/v1/raInfo | python -m json.tool
```


blueplanet

Lab: Render a Custom RA Project

- Complete the following lab:
  - Lab 7: Create Custom RA Project

Notify your instructor when you have completed these labs.

blueplanet

Blue Planet Resource Adapter (RA) Development

Agenda

1. Blue Planet and RA Development Overview
2. Creating an RA Project
3. bpprov Overview
4. Creating a CLI-Based RA
5. Customizing an RA

blueplanet

blueplanet
a division of ciena

Customizing an RA

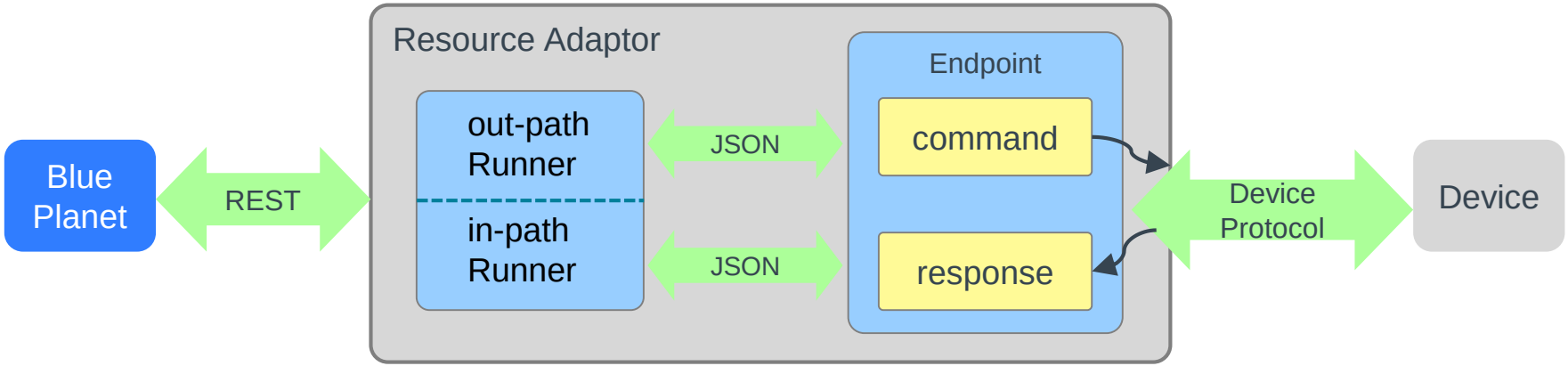
Customizing an RA - Preview

- Topics covered in this section include:
  - How to build command files
  - The **show-device.json** command file

blueplanet

Building Commands

- Commands are defined in JSON files
- Most work is building Runners (which transforms data into other structures)
- Commands go in the ***RA/raconf*/model/commands** directory



bpprov Command Lookup

- Commands should be separated by device into sub-directories.
- Commands common to multiple devices go in the ***RA*/*raconf*/model/commands** directory.
  - If the command is only in one place, it will be used.
  - Same command can be in common directory and in family or device directory
  - If the command is in multiple locations, device location is used first, then family, then ***RA*/*raconf*/model/commands**.
- **show-device.json** command file ***must*** be in ***RA*/*raconf*/model/commands**.
  - **show-device.json** is used to return the device's RESOURCE_TYPE and DEVICE_TYPE
  - Must be used for all devices

show-device Command

- **show-device.json** command file is required
  - A response to **show-device.json** command must contain attributes which map directly to the values in **family.json** and **device.json**
- Output complies to the schema in ***RA/raconf*/model/schema/in-show-device.json** (which points to **models/Device.json**)

```
{
    "type": "<DEVICE_TYPE>",
    "resourceType": "<RESOURCE_TYPE>",
    "serialNumber": "2837982731",
    "deviceVersion": "2.3.4c",
    "swType": "demoOS",
    "swImage": "lastest-and-greatest",
    "swVersion": "4.5.6.7s"
}
```

Must match one of the DEVICE_TYPE names from the **device.json** file.

Must match one of the RESOURCE_TYPE names from the **family.json** file.

May need to match version value from the **device.json** file.

bpprov Command File Format

```
{
    "endpoint": "<endpoint>",
    "type": "<runner>",
    "endpoint-parameters": {
        "command": "<command text>"
    },
    "out-path": [ {
         "type": "<translator>”,
         "parameters": { }
         }
    ],
    "in-schema": "<schema file>",
    "in-path": [{
         "type": "<translator>",
         "parameters": { }
         }
    ]
}
```

<endpoint> must match one of the ENDPOINTS defined in **settings.py**

Three <runner> types; discussed later in this course

<commands text> will be the commands to execute on the device

out-path are data transformations made to data coming from BPO

in-schema is the Schema file used to verify the final output of in-path data

in-path are data transformations made to data coming from device

blueplanet

Lab: Create show-device.json

- Complete the following lab:
  - Lab 8: Create show-device.json

Notify your instructor when you have completed these labs.

blueplanet

TextFSM

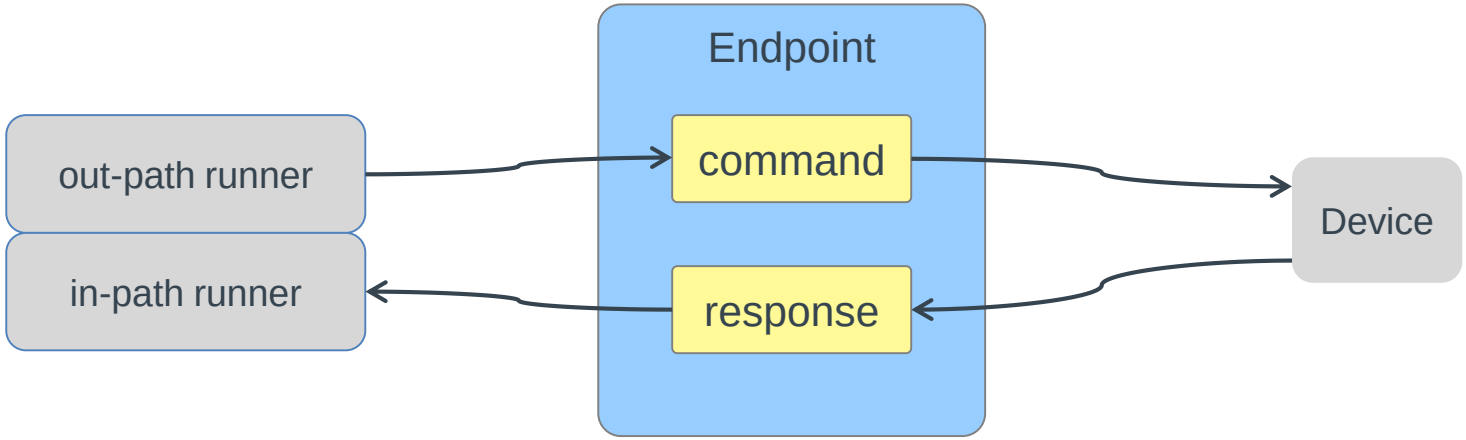
TextFSM - Preview

- Topics covered in this section include:
  - An introduction to TextFSM
  - TextFSM template files
  - Value definitions
  - State definitions
  - bpprov-cli fsm-validate

blueplanet

Endpoint Command

- Specifies the command to be sent to the device
- Result of out-path can be used as the command
  - out-path runs before command is executed



blueplanet

Building show-device.json Output

- In lab, two commands will be used:
  - **uname -snrmpo**
    ```
    Linux vagrant 4.4.0-51-generic x86_64 x86_64 GNU/Linux
    ```
  - **docker -v**
    ```
    Docker version 1.12.6-cs13, build 0ee24d4
    ```
- Goal will be to produce the following output:

```
{
    "swImage": "GNU-Linux",
    "swVersion": "4.4.0-51-generic",
    "resourceType": "docker",
    "serialNumber": "vagrant",
    "swType": "Linux",
    "type": "linux",
    "deviceVersion": "1.12.6-cs13, build 0ee24d4"
}
```


blueplanet

TextFSM

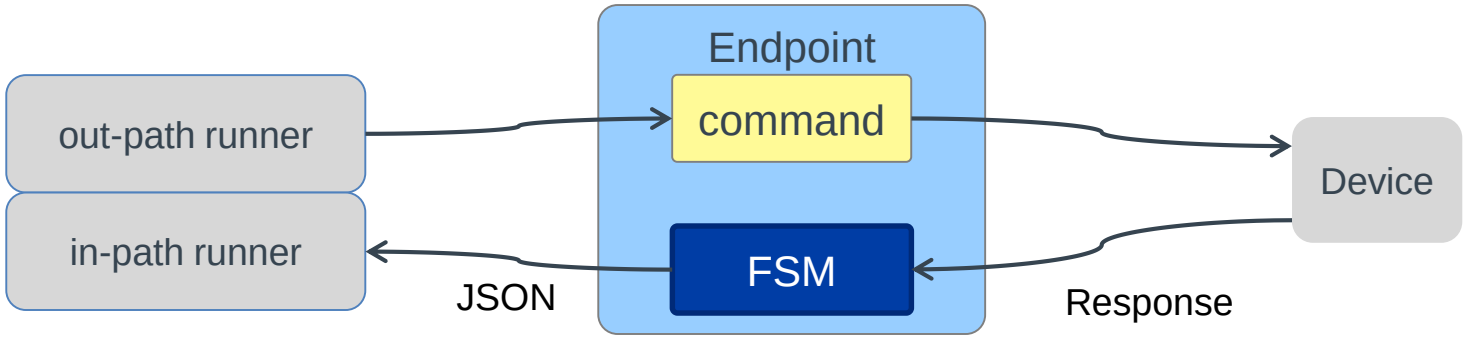
- Endpoint process response using TextFSM
- TextFSM:
  - Open source Python package
  - https://code.google.com/archive/p/textfsm/wikis/TextFSMHowto.wiki
  - Additional documentation: https://github.com/google/textfsm/wiki/Code-Lab
  - Implements a template based state machine for parsing semi-formatted text, i.e. CLI response
- Inputs:
  - Template file
  - Text input – response from the device
- Template files go in ***RA/raconf*/model/fsms**

blueplanet

TextFSM

- FSM is optional (not always needed)
- FSM is part of the endpoint:

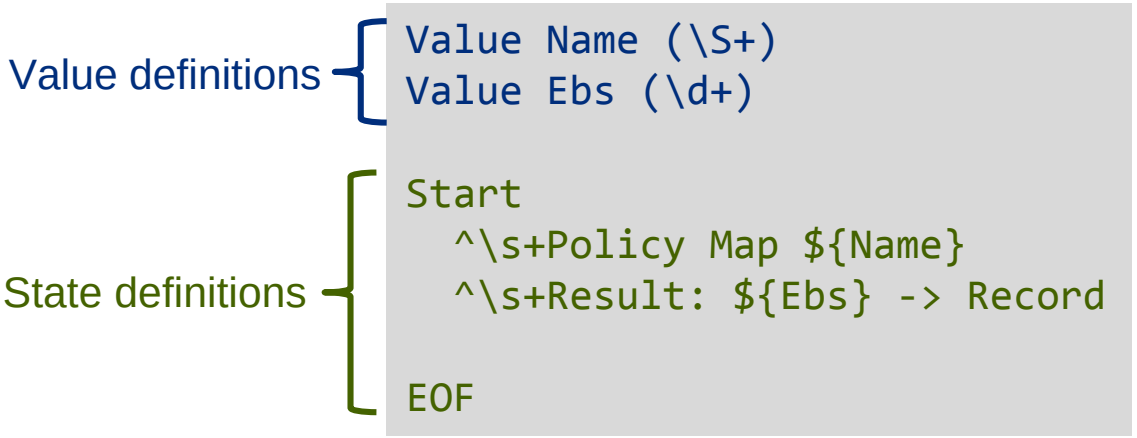
```
"endpoint-parameters": {
    "command": "uname -snrmpo",
    "fsm": "fsms/show-device.fsm"
}
```



blueplanet

FSM Template File

- TextFSM template file has two main sections:
  - Value definitions – defines the fields and types that should be extracted
  - State definitions – defines the rules for how to parse the content



blueplanet

Value and State Definitions

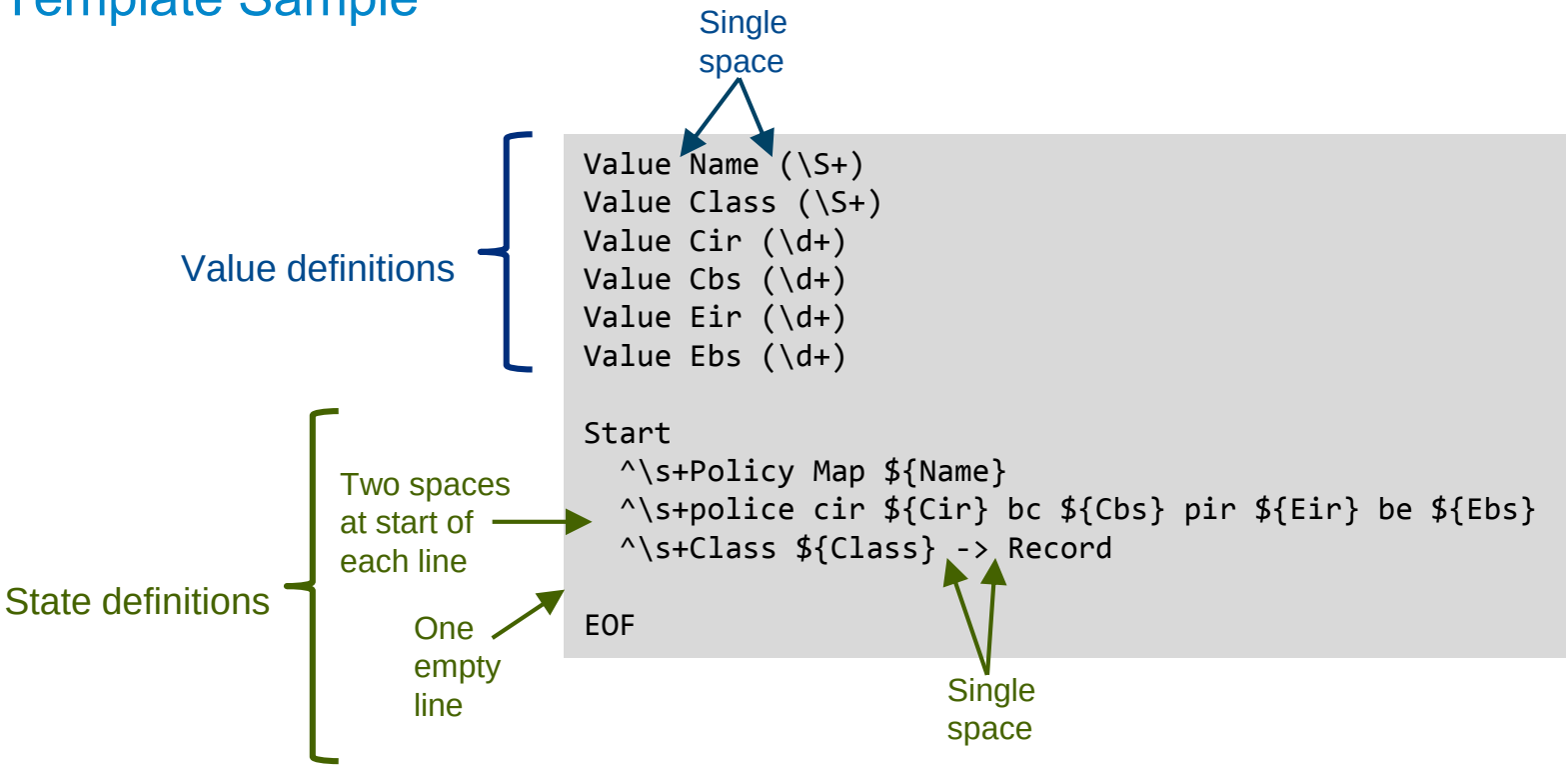
- Value Definitions:
  - Regex defines valid characters to be filled in for the value:

    `Value ERPName (regex)`
  - Only the Value Definitions are stored in Record
- State Definitions:
  - Start newline with **two spaces** followed by `^`
  - End last definition only with `-> Record`
  - Format and spacing must match “record format” of input text
  - Use a combination of Value Definitions, text and regex to match data:

Input Text:	`1    Training ERP                    600-650`
State Definition:	`^${Ind}\s+${ERPName}\s+${VIDSet} -> Record`

blueplanet

FSM Template Sample



blueplanet

Using bpprov-cli fsm-validate

- Used to test a FSM template without having to add it to a command file
- Example:

```
env/bin/bpprov-cli fsm-validate RA/raconf/model/fsms/test.fsm
--input=@fsm_test.txt
```

- **test.fsm** is the FSM template file
- **fsm_test.txt** is sample output from a device

blueplanet

Lab: Create an FSM Template

- Complete the following lab:
  - Lab 9: Create FSM

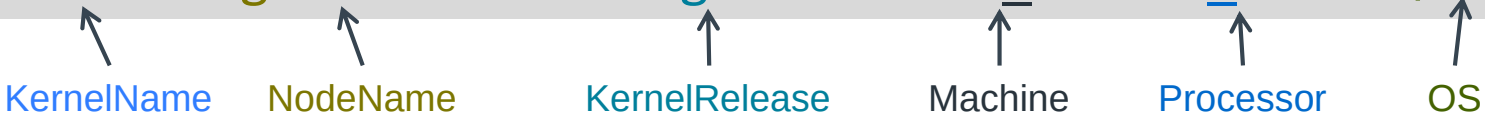
Notify your instructor when you have completed these labs.

blueplanet

show-device.fms

- Create an FSM template to parse the result of **`uname -snrmpo`**

```
Linux vagrant 4.4.0-51-generic x86_64 x86_64 GNU/Linux
```



blueplanet

Lab: Add an FSM Template to show-device

- Complete the following lab:
  - Lab 10: Add an FSM Template to show-device

Notify your instructor when you have completed these labs.

blueplanet

blueplanet
a division of ciena

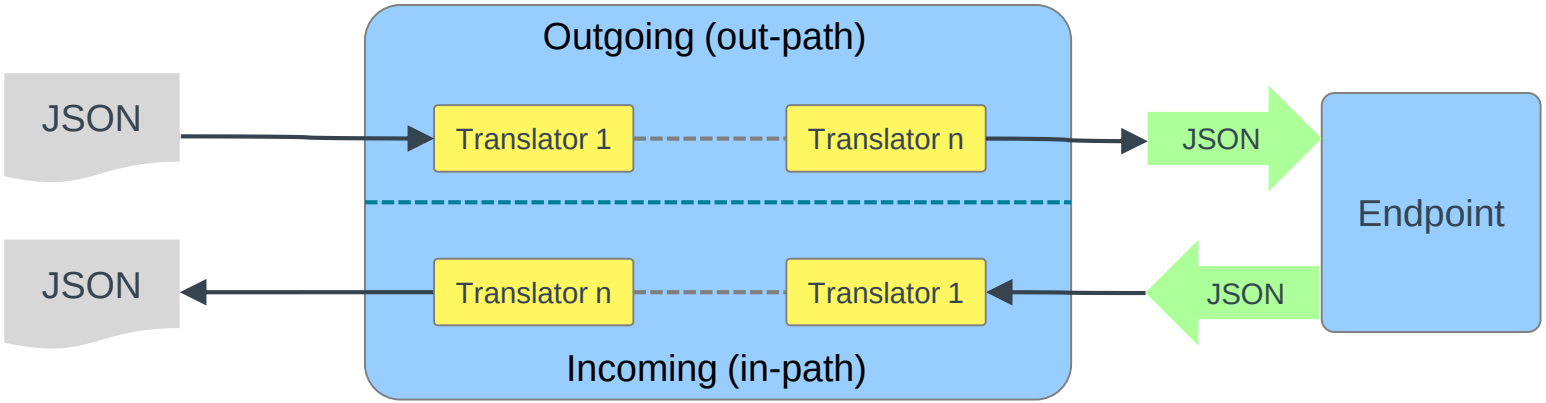
Runners and Translators

Runners and Translators - Preview

- Topics covered in this section include:
  - Review of Runners
  - Types of Runners
  - Introducing Translators:
    - bpprov.translators.list.Flatten
    - bpprov.translators.list.ToDict
    - bpprov.translators.template.Json
    - bpprov.translators.aggregator.Call

Runners

- bp-prov commands contain two paths: **in** and **out**
- Each path has a runner
  - Runner is set of translators that manipulate data



blueplanet

Types of Runners

- simple.Event
  - Translation runner for simplex asynchronous data.
- simple.Sequence
  - Translation runner for bidirectional data against a given endpoint.
  - Execution runs in two stages: out-path and in-path.
  - Translators run in the order that they appear in the path.
- simplepy.PySequence
  - Python runner for bidirectional execution against any endpoints.
  - The runner loads the script and runs the Class.execute method.
  - A blocking call to any endpoint can be made with Class.execute_ep()
    - Works much like a normal simple sequence .json runner.

Translators

- bpprov has 50+ built in translators
  - All are documented in bpprov docs
- Frequently used in class (note: these are case-sensitive):
  - bpprov.translators.list.Flatten
    - Flattens a nested list into a single list
    - Use to flatten output of TextFSM
  - bpprov.translators.list.ToDict
    - Converts a list into a dictionary
    - Requires an array of labels
  - bpprov.translators.template.Json
    - Translate incoming structure into JSON structure, based on a template

Syntax for Adding Runners

- in-path and out-path are arrays
  - Each item in the array is an object with values like:
    - **type** specifies the translator
    - **parameters** specify optional information need to process data
- Example:

```
"in-path" : [{
    "type": "bpprov.translators.list.Flatten"
},{
     "type": "bpprov.translators.list.ToDict",
     "parameters": {
         "labels": [ "dockerVersion", "dockerBuild" ]
     }
 }]
```


blueplanet

Lab: Add in-path Runner

- Complete the following lab:
  - Lab 11: Add in-path Runner

Notify your instructor when you have completed these labs.

blueplanet

Combining Commands

- bpprov.translators.aggregator.Call
  - Calls another bpprov command file
  - “command” parameter is required
  - Output **patches** includes an array of JSON Pointer patches to move the data

blueplanet

bpprov.translators.aggregator.Call

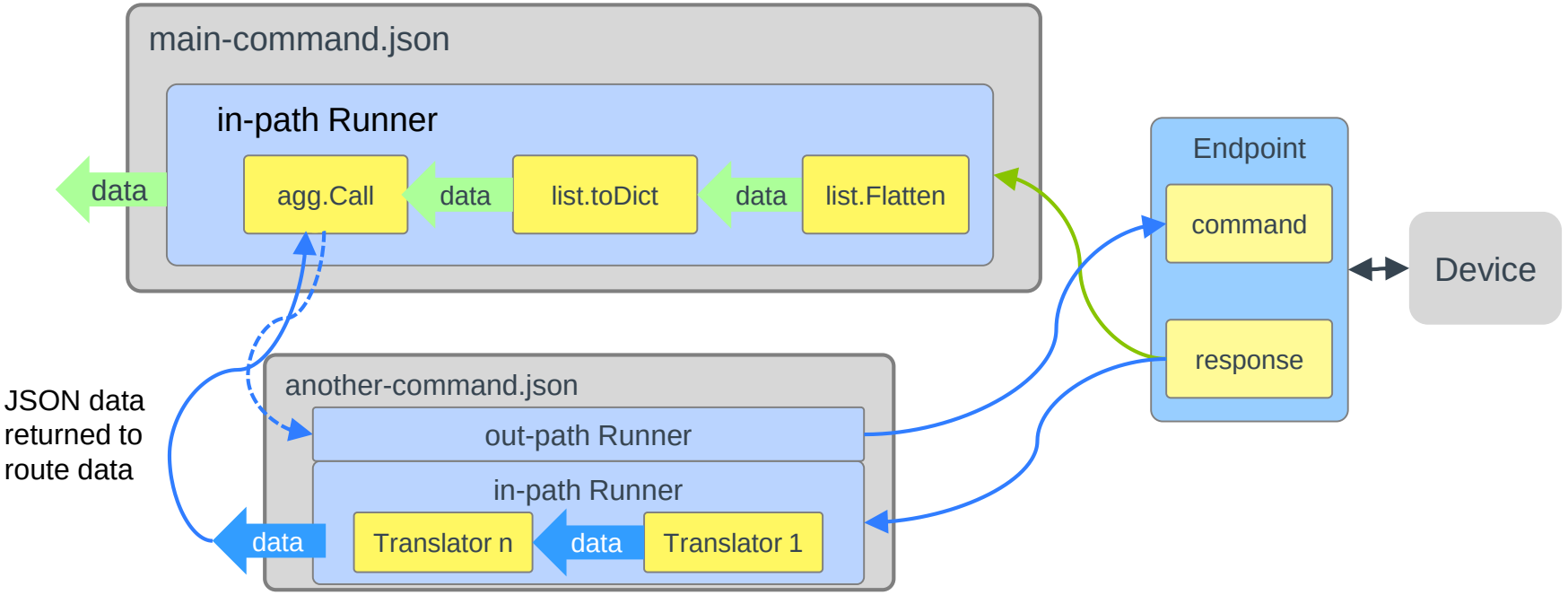
```
"type": "bpprov.translators.aggregator.Call",
        "parameters": {
            "command": "/commands/show-docker-version.json",
            "output": {
                "patches": [
                    {
                        "from": "/data/dockerVersion",
                        "to": "/data/dockerVersion"
                    },
                    {
                        "from": "/data/dockerBuild",
                        "to": "/data/dockerBuild"
                    }
                ]
            }
        }
```

from data is JSON data sent from command output

to data is JSON route data in runner

blueplanet

Aggregator Call Example



blueplanet

Lab: Create show-docker-version.json

- Complete the following labs:
  - Lab 12: Create show-docker-version.json
  - Lab 13: Add a Command Aggregator

Notify your instructor when you have completed these labs.

blueplanet
a division of ciena

Jinja2

Jinja2- Preview

- Topics covered in this section include:
  - Introduction to Jinja2
  - Hardcode show-device.json

blueplanet

Render Template

- bpprov.translators.template.Json
  - Translates incoming structure into a JSON structure based on a template
  - Jinja2 is used to process templates
- Template Types:
  - **file**: Retrieve the template from file specified in "template" (default)
  - **inline**: Retrieve the template from the string that represents a valid JSON structure
  - **json-dict**: Retrieve the template from the JSON structure specified in "template". Each key and value of the dictionary will be treated as a template
- Store **.tmpl** files in ***RA*/*raconf*/model/templates**
- Use {{data.field}} to insert data from runner into template

blueplanet

Render Template Example

Runner:

```
"type": "bpprov.translators.template.Json",
        "parameters": {
            "template": "show-device.tmpl",
        }
```

Template example:

```
{
    "type": "docker",
    "resourceType": "NixDockerServer",
    "serialNumber": "{{ data.node }}",
    "deviceVersion": "{{ data.dockerVersion }}, build {{data.dockerBuild}}",
    "swImage": "{{ data.kernel }}",
    "swType": "{{ data.processor }}",
    "swVersion": "{{ data.release }}"
}
```


blueplanet

Hardcode show-device.json

Not all RAs need complex **show-device.json**

- An RA that only communicates with a single device always returns same output when **show-device.json** is run
- Solution: hardcode the output

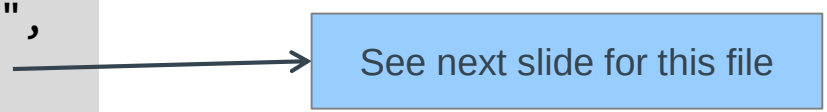
Several techniques to hardcode:

- Run a “fake” command on the device (not ideal)
- Use a “null” endpoint (better solution)

blueplanet

Hardcode show-device.json – null endpoint technique

```
{
    "endpoint": "null",
    "type": "bpprov.runners.simple.Sequence",
    "out-path": [
    ],
    "in-schema": "in-show-device.json",
    "in-path": [{
        "type": " “
        } bpprov.translators.template.Json",
        "parameters": {
            "template": "show-device.tmpl
    }]
}
```



blueplanet

Hardcode show-device.json – the template file

```
{
    "type": "linux",
    "resourceType": "docker",
    "serialNumber": "123456789",
    "deviceVersion": "1.0.0",
    "swType": "Example",
    "swImage": "Rev 1",
    "swVersion": "1.0.0"
}
```


blueplanet

Hardcode show-device.json – alternative method

```
{
    "endpoint": "null",
    "type": "bpprov.runners.simple.Sequence",
    "out-path": [
    ],
    "in-schema": "in-show-device.json",
    "in-path": [{
        "type": "bpprov.translators.importer.JsonContents",
        "parameters": {
            "file": "data/example-device.json"
        }
    }]
}
```

Imports contents of JSON file into the data path. Doesn't not attempt to run any Jinja operations.

Contains the same data as template file on previous slide

blueplanet

Using a null endpoint

Put in both "endpoints" files (**endpoint-params.json** and **endpoints.json**):

```
"null": {
      "name": "null",
      "type": "bpprov.components.null.NullEndpoint",
      "parameters": { }
  }
```

And use this command:

```
env/bin/bpprov-cli command-run RA/raname/model null
commands/show-device.json '{}' --start
```

Use “null” here instead of “cli”

blueplanet

Lab: Create complete show-device.json Command

- Complete the following lab:
  - Lab 14: Complete show-device.json

Notify your instructor when you have completed these labs.

blueplanet

blueplanet
a division of ciena

Troubleshooting

Troubleshooting - Preview

- Topics covered in this section include:
  - Troubleshooting features of **`bpprov-cli`**
  - BPPROV Pipeline Visualizer Debug Tool
  - Troubleshooting tactics and tips

blueplanet

Additional bpprov-cli features

- **`env/bin/bpprov-cli -h`**
  - fsm-validate Validate FSM template
  - validate-json Validate JSON files
  - command-run Run a specific command against a specific endpoint.
  - validate Validate configurations

blueplanet

Validate JSON with bpprov-cli

- **`bpprov-cli validate-json <directory>`**
- All files in that directory are validated
- No output means no errors found
- Might issue errors for irrelevant things (like tab characters)
- Example:

```
$ env/bin/bpprov-cli validate-json radocker/model/commands
Traceback (most recent call last):
Ignore the Python error messages
ValueError: Invalid JSON file 'radocker/model/commands/show-device.json',
error: 'Expecting object: line 49 column 2 (char 1339)'
```

Validate entire project

- **`bpprov-cli validate <directory>`**
- Checks several components:
  - File structure & contents
  - Missing commands, fsms, templates

blueplanet

bpprov-cli validate successful example

```
$ env/bin/bpprov-cli validate radocker/model
Running validation:
  Run validation on device model
  Global validation
  Run validation on JSON files
  Make sure all files follow certain extension
  -- commands
  -- fsms
  -- idmappers
  -- templates
  Make sure all runners have the correct syntax/format
```

bpprov-cli validate unsuccessful example

```
$ env/bin/bpprov-cli validate radocker/model
Running validation:
  Run validation on device model
<output omitted>
packages/bpprov/cli/commands/validate.py", line 189, in run
    result |= validator(self.config, self.args).run()
  File "/home/vagrant/shared/radocker/env/local/lib/python2.7/site-
packages/bpprov/cli/commands/validate.py", line 81, in run
    self._assert_true(device_d['family'] in family_d, "Invalid family
'{}'".format(device_d['family']))
  File "/home/vagrant/shared/radocker/env/local/lib/python2.7/site-
packages/bpprov/cli/commands/validate.py", line 64, in _assert_true
    raise ValueError(msg)
ValueError: Invalid family 'NixDockerServer'
```

Debugging commands

- Drop **--start** option to enter a debugger mode
- Allows you to step through operations one at a time
- Note: ' {}' is used to send data via out-path

```
$ env/bin/bpprov-cli command-run radocker/model cli commands/show-device.json '{}'
Runner: commands/show-device.json
Path   : start out:0
{}
bpdb$ h
bp-prov debugger
  b, break              add breakpoint
  bl, breaklist         list all breakpoints
  c, continue           until the next breakpoint
  cl, clear             one or all breakpoints
  {remaining output omitted}
```

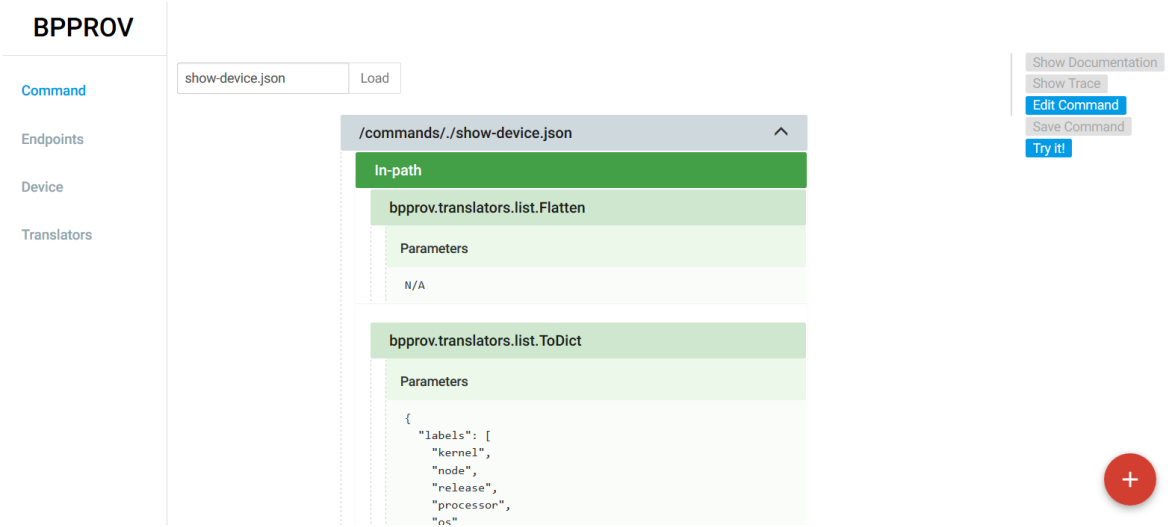

blueplanet

Debugger: BPPROV Pipeline Visualizer Debug Tool

- Web-based Server
- Do not need to run RA – just works on bpprov pipeline

Start Debugger: `env/bin/bpprov-ui <ra name>/model/`

Access Debugger: http://192.168.33.10:9123/ui/



blueplanet

Troubleshooting 101

- **Always back up configuration files before making changes**
- Look closely at the error messages
- Understand that Python message:
  - Don’t indicate a problem with Python
  - Indicate a problem with your configuration
- Use validators
  - http://jsonlint.com/ - for individual JSON files
  - **`bpprov-cli fsm-validate`**
  - **`bpprov-cli validate-json`**
  - **`bpprov-cli validate`**

blueplanet

Lab: Resolving Errors

- Complete the following lab:
  - Lab 15: Resolving Errors

Notify your instructor when you have completed these labs.

blueplanet
a division of ciena

Additional commands

Additional commands - Preview

- Topics covered in this section include:
  - Completing the RA by adding additional CRUDL commands
  - The bpprov.translators.call.Function translator
  - Further details regarding Jinja2

blueplanet

Additional Commands

- **show-containers.json**

```
docker ps -a -q | xargs docker inspect
```

  - Command response is a JSON object
  - Use jinja template to return sub-set of command response
- **start-container.json**

```
docker run {{ data.parameters }} {{ data.image }} {{ data.command }}
```

  - Data from Orchestrate will be used in command
- **remove-container.json**

```
docker rm -f {{ data.container }}
```

  - Data from Orchestrate will be used in command

blueplanet

show-containers.json Command

- Structure:
  - Send the command: **`docker ps -a -q | xargs docker inspect`**
  - No FSM need
  - in-path:
    - Call *json.loads* using bpprov.translators.call.Function
      - *json.loads* is a Python function that converts JSON formatted data into a JSON object
    - Parse with Jinja using bpprov.translators.template.Json

blueplanet

Call Function

- bpprov.translators.call.Function
  - Call a Python function to transform data
- Function is addressed by the dot module notation
- Function should not maintain a state (e.g. global variable) since the module might be reloaded
- Function should have at least one argument for the data, and optional **kwargs to pass "params”

Jinja

- Control Structures
  - if, for
  - control structures appear inside {% ... %} blocks
  - http://jinja.pocoo.org/docs/dev/templates/#list-of-control-structures
- Filters
  - Separated from the variable by a pipe symbol (|)
  - http://jinja.pocoo.org/docs/dev/templates/#builtin-filters
  - Code fragment:

```
{% for record in data %}
  "user": "{{ record.username|e }}"
{% endfor %}
```

Jinja example

```
[
{% for flavor in data.flavors %}
{
    "ID": "{{ flavor.id }}",
    {% if flavor.name is defined %}
        "name": "{{ flavor.name|lower }}",
    {% endif %}
    {% if flavor.ram is defined %}
        "ram": "{{ flavor.ram }}",
    {% endif %}
}
{% endfor %}
]
```

Mapping Properties to Orchestrate

container.Name

container.State.Running

container.NetworkSettings
.IPAddress

container.Image

container.Id

container.Path, container.Args

container.Config.Image

container.Name

container.State.Running

```
{
    "label": "bpocore-dev",
    "productId": "5813c5cf-5b7c-4055-ab64-19299dc0b611",
    "tenantId": "7c7da24e-d2af-35c7-8f6d-d8d16c7f0738",
    "properties": {
        "status": "Up",
        "ipAddress": "172.17.0.2",
        "imageId": "490472e4fb4fd3171254619c156bff054883afac2d3f8bc3a648318e8d54dfdb",
        "id": "47f5119388abbf3d0ed9e4c1e0ff456fb4a90eb7b0046005e80cdfea201df34c",
        "command": "/bin/sh -c /main.py",
        "image": "5813c5d0-0c43-416a-afa3-3f73f417a9e0",
        "name": "/bpocore-dev"
    },
    "providerResourceId": "47f5119388abbf3d0ed9e4c1e0ff456fb4a90eb7b0046005e80cdfea201df34c",
    "discovered": true,
    "orchState": "active",
    "reason": "",
    "id": "5813c5d0-dc9f-4261-8cff-49a4562f3395",
    "resourceTypeId": "nixtraining.resourceTypes.nixContainer",
    "shared": false,
    "differences": [],
    "desiredOrchState": "unspecified",
    "tags": {},
    "providerData": {},
    "updatedAt": "2016-10-28T21:40:34.336Z",
    "createdAt": "2016-10-28T21:40:32.925Z",
    "autoClean": false
}
```

Container properties from docker

blueplanet

Lab: Create show-containers.json Command

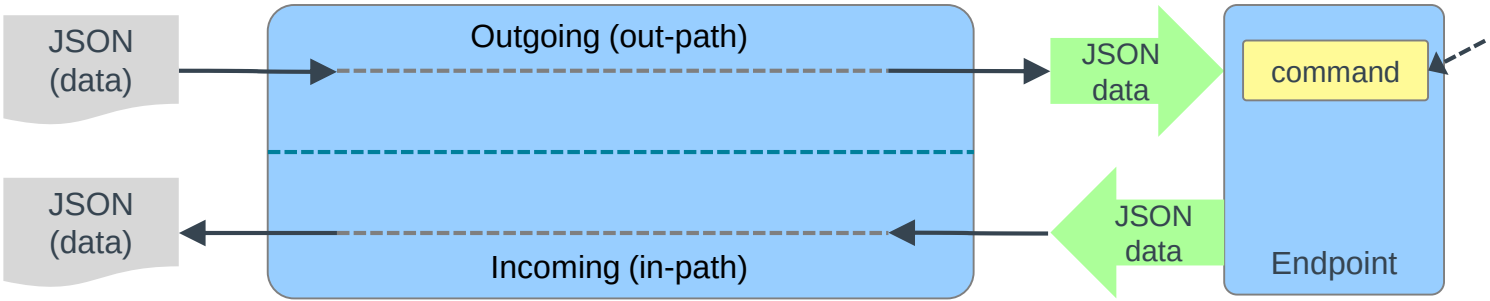
- Complete the following lab:
  - Lab 16: Create show-containers.json

Notify your instructor when you have completed these labs.

blueplanet

start-container.json Command

- Command structure:
  - Sends command: **`docker {{ data.parameters }} {{ data.image }} {{ data.command }}`**
  - No FSM needed
  - No in-path or out-path runners needed
    - Leave empty arrays in the command file
  - Route data is passed to endpoint and inserted into CLI command



data from JSON is inserted into CLI command

blueplanet

Lab: Create start-container.json Command

- Complete the following lab:
  - Lab 17: Create start-container.json

Notify your instructor when you have completed these labs.

blueplanet

remove-container.json Command

- Command structure:
  - Sends command: **docker rm -f {{ data.container }}**
  - No FSM needed
  - No in-path or out-path runners needed
    - Leave empty arrays in the command file
  - Container name is passed to endpoint to insert into CLI command

Labs

- Complete the following labs:
  - Lab 18: Create remove-container.json
  - Lab 19: Test Your RA

Notify your instructor when you have completed these labs.

blueplanet
a division of ciena

End of Part 1