blueplanet

Blue Planet

Multi-Service Domain Orchestration

C-808 DevOps Toolkit Software Skills
Rev 3
Blue Plane Release 18.06.4

Blue Planet Orchestration DevOps Software Skills

Agenda

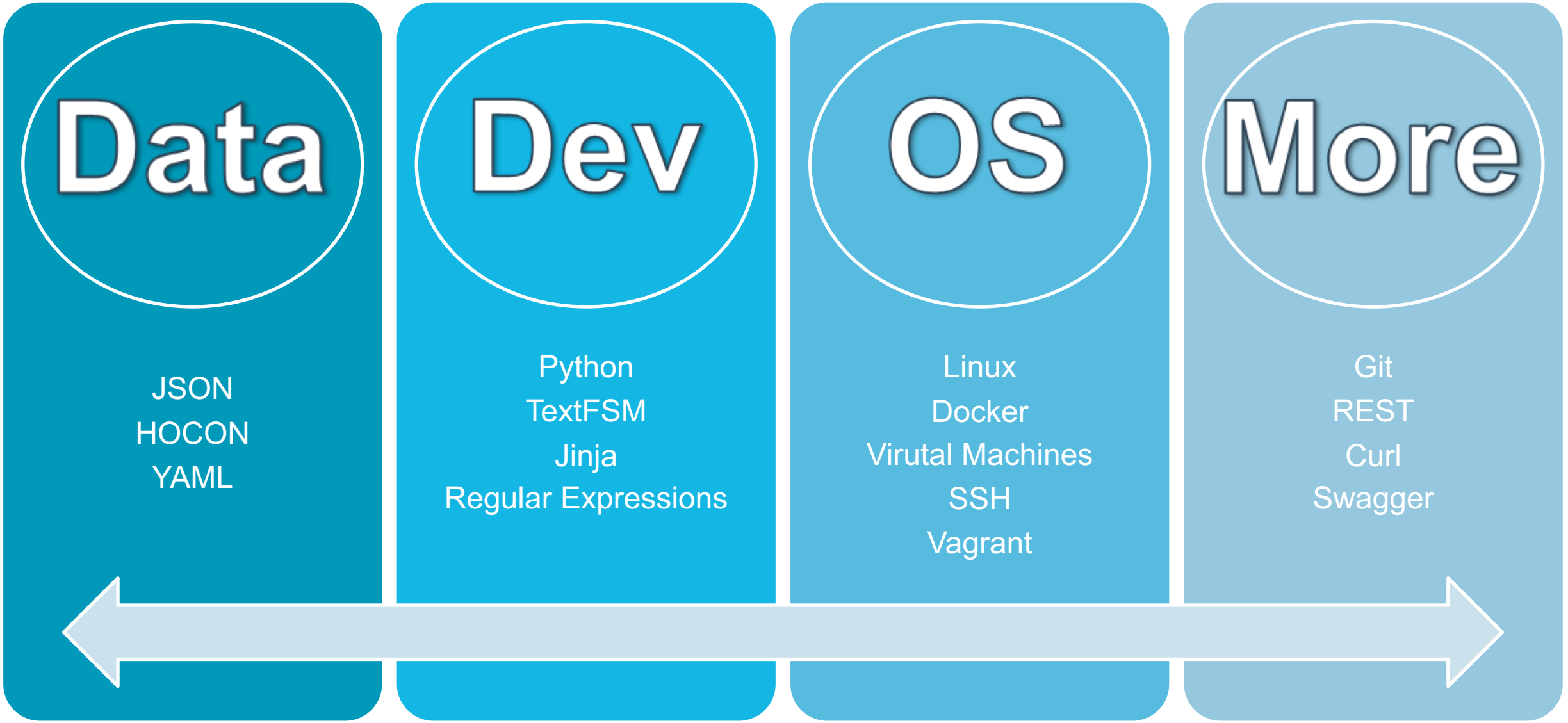
1	**Overview of Software Tools used in Blue Planet**
2	Data Structures
3	Introduction to Devops Toolkit
4	Python Programming

blueplanet®

Important Note

- **This class is designed to provide learners with basic software skills necessary to attend the following courses:**
  - C-802: Blue Planet Orchestrate DevOps Toolkit Service Onboarding
  - C-807: Blue Planet Orchestrate DevOps Toolkit Resource Adapter Development
- **This course does not teach specific Blue Planet Orchestrate topics, but rather topics that you must know in order to develop in Blue Planet Orchestrate**
- **In order to understand how these topics relate to Blue Planet Orchestrate, the following terms are used:**
  - Blue Planet – Refers to the Blue Planet Orchestrate software
  - Resource Adapter – Refers to the software that communicates with devices

Blue Planet Toolbox



blueplanet

Data Structures - Overview

- **Data in Blue Planet needs to be transferred between different components**
- **Configuration files also need a well-definded data structure**
- **JSON**
  - The primary configuration format for Resource Adaptors (Resource Adapters), the component of Blue Planet that communications with devices
  - Also used to transfer data between components
- **HOCON – The primary configuration structure for Data Modeling, the process of gathering data to send to the Resource Adapter**
- **YAML – Used for some additional configuration files, such as the Resource Provider (how Resource Adapters communication with Blue Planet)**

Software Development - Overview

- **Most code is written in Python**
- **Knowing Python is useful, but not necessarily required**
- **TextFSM – A Python-based tool for converting plain text into JSON**
- **Jinja – A Python-based tool used to perform complex data manipulation**
- **Regular Expressions**
  - Special characters designed to perform pattern matching operations
  - Used in many different area of Blue Planet development
  - A feature of Python, TextFSM and Jinja

Operating Systems - Overview

- **Linux: Standard Blue Planet host operating system**
  - Ubuntu – Used in the Devops TK
  - Red Hat Enterprise Linux – Standard host for production Blue Planet servers
- **Virtual Machines (VMs)**
  - Ability to run a different operating system within a host operating system
  - Used in the Devops TK
  - Examples: Virtual Box and Vmware (Virtual Box is standard Devops TK)
  - Vagrant is a VM-based tool that makes it easy to configure an existing VM image
- **Docker – similar to VMs, but more light-weight**
- **SSH – A protocol that allows command-line connections to VMs or remote systems**

Additional Tools - Overview

- **GIT – A tool designed to maintain different versions of files**
- **REST – A protocol designed to communicate between software**
  - Used to communicate to an Resource Adapter or to Blue Planet server
  - **curl**: a command-line tool designed to send REST-based data
  - Swagger: a web-based tool designed to send REST-based data

Blue Planet Orchestration DevOps Software Skills

Agenda

1	Overview of Software Tools used in Blue Planet
2	**Data Structures**
3	Introduction to Devops Toolkit
4	Python Programming

blueplanet

What is JSON?

- **JSON – JavaScript Object Notation**
  - http://json.org/
  - Data-interchange language for programming
  - Text format that is language independent
  - Uses UTF-8 text
- **JSON is built on two structures:**
  - A collection of name/value pairs
  - An ordered list of values
- **JSON Validator: http://jsonlint.com/**

blueplanet

JSON Objects

- **An *object* is an unordered set of key/value pairs**
  - Begins with **{** (left brace)
  - Ends with **}** (right brace)
  - Trailing commas are not permitted
- **Example:**

```
{
    "make": "Ford",
    "model": "Mustang",
    "year": 1965
}
```

- **Note: key/value pairs are not stored in a specific order**

blueplanet

JSON Array

- **An *array* is an ordered collection of values.**
  - Begins with **[** (left bracket)
  - Ends with **]** (right bracket)
  - Values are separated by , (comma)
  - Trailing commas are not permitted
- **Example:**

```
{
    "make": "Ford",
    "model": "Mustang",
    "year": 1965,
    "color": ["Blue", "Black", "White", "Red"]
}
```


blueplanet

JSON Values

- **String**
  - Value in double quotes: `"Blue"`
  - Backslash (\ )for escape:
    - `"\"Blue\""` results in `"Blue"`
- **Number**
- **Object**
- **Array**
- **True or False**
- **Null**

blueplanet

HOCON Introduction

- **HOCON – Human-Optimized Config Object Notation**
- **Primary goal:**
  - Keep the semantics (tree structure; set of types; encoding/escaping) from JSON, but make it more convenient as a human-editable config file format.
- **HOCON is more flexible than JSON**
- **All valid JSON remains valid**
- **Invalid files will generate an error**
- **HOCON Validator: http://www.hoconlint.com**
- **Reference: GitHub**

blueplanet

HOCON Features

- **Less noisy, cleaner syntax**
- **Ability to refer to another part of the configuration (set a value to another value)**
- **Import or include another configuration file into the current file**
- **Commenting**

blueplanet

HOCON Syntax

- **Omit root braces**
- **Omit quotes on keys, use quotes for strings**
- **Can use = instead of : for key/value separator**

```
make = "Ford"
```

- **Don’t need a comma if there is a newline**
- **Key can be followed by { to create objects.**

```
car { make = "Ford"
      model = "Mustang"}
```


blueplanet

HOCON Syntax

- **Characters outside of quotes are parsed as strings if:**
  - There are no forbidden characters (`{`, `}`, `[`, `]`, etc.)
  - Doesn't contain `//` or `#`
  - Initial character doesn't parse a `true`, `false`, `null` or a number
    - `1965 Ford Mustang` would not be a valid string (requires quotes)
  - Unquoted strings are combined into a single strings
    - `Ford Mustang` becomes `"Ford Mustang"`
- **Use triple quotes (`"""`) to start and end multiline strings**
  - Every thing between sets of `"""` is treated as text, even forbidden characters, newlines, and whitespace

blueplanet

HOCON/JSON Comparison

- **JSON**

```
 1  { "car":
 2      {
 3          "make" : "Ford",
 4          "model" : "Mustang",
 5          "year" : 1965,
 6          "color" : [
 7              "Blue",
 8              "Black",
 9              "White",
10              "Red"
11          ]
12      }
13
14  }
```

- **HOCON**

```
 1  car {
 2      make = Ford
 3      model = Mustang
 4      year = 1965
 5      color = [
 6          Blue
 7          Black
 8          White
 9          Red
10      ]
11  }
12
```


blueplanet

JSON Schema

- Describes the valid format of specific JSON and HOCON files
- Critical to understand when building Resource Adapters
- Important: Must know how to read JSON Schema (not how to write one)
- Resource: http://json-schema.org/
- Schema Validator: http://www.jsonschemavalidator.net/

blueplanet

JSON Schema Example

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

Define the properties

No other properties allowed

Must have these properties

blueplanet

JSON Schema Properity Types

- **string**
  - Value should be in quotes
- **number**
  - Value not in quotes
  - Can also use integer if supported
- **boolean**
  - Value can be true or false (not in quotes)
- **array**
- **object**
  - This can contain anything that the overall JSON object can contain
  - Nested objects are allowed (and common)

blueplanet

String Properity Parameters

- **Additional parameters are used to fine tune string values:**
  - pattern – Used to specify a Regular Expression that the value must match
  - oneOf – Specify a sub-type that the value must match, example:

```
“host": {
        "type": "string",
        "oneOf": [
            { "format": "host-name" },
            { "format": "ipv4" },
            { "format": "ipv6" }
        ]
    }
```


blueplanet

Number or Integer Properity Parameters

- **Additional paramaters are used to fine tune number values:**
  - minimum – Specifiy the minimum value
  - maximum – Specify the maximum value

```
“RAM": {
        "type": "integer",
        "minimum": 16,
        "maximum": 512
    }
```


blueplanet

Array Properity Parameters

- **Additional paramaters are used to fine tune array values:**
  - items – An object that indicates the type of items allowed in the array
  - minItems – The minimum number of items that must be in the array
  - uniqueItems – A Boolean value that specifics that each value of the array must be unique

```
“list": {
        "type": "array",
        "minItems": 1,
        "items": { "type": "string" },
        "uniqueItems": true
    }
```


blueplanet

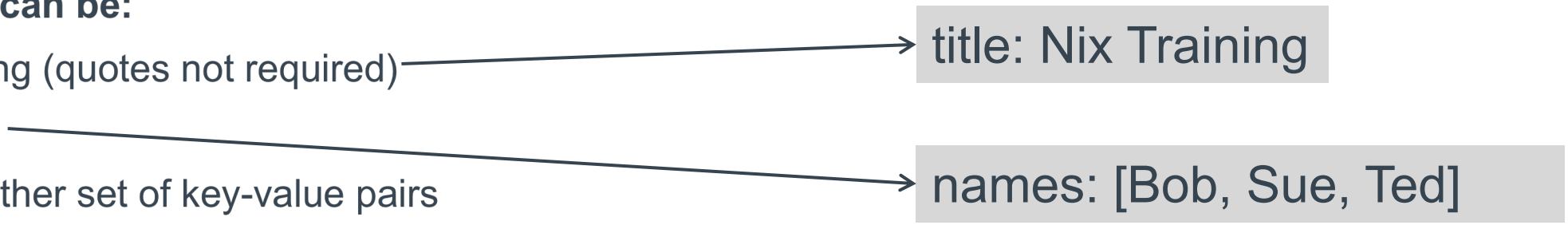
What is YAML

- **YAML– YAML Ain't Markup Language**
  - **http://yaml.org/**
  - human friendly data serialization standard
  - Uses similar structure to JSON
  - Comments permitted (use # character)
- **YAML Validator: http://www.yamllint.com/**

blueplanet

YAML Key-Value Pairs

- **Keys do not need to be in quotes**
- **Values can be:**
  - String (quotes not required)
  - List
  - Another set of key-value pairs
    - AKA sets or mappings



blueplanet

YAML Mapping Example

- **Primary key**
- **Secondary keys**
- **Ternary keys**

- **Indentation provides structure**
- **Curly braces can also be used:**
  - properties: {

    name: username,

    title: "Username“

    }, {

    name: password

    title: "Password“

    }

```
domains:
  - id: "urn:cienatraining:bp:domain:ranixtraining"
    title: "Docker"                #Comments allowed
    properties:
      - name: username
        title: "Username"
        optional: false
        type: string
      - name: password
        title: "Password"
        optional: false
        obfuscate: true
        type: string
```


blueplanet

Text Editing Tools

- **On Linux - Typically only have command-line editors:**
  - **`vi`** or **`vim`**
  - **`joe`** (may need to be installed)
  - **`emacs`** (may need to be installed)
- **On Windows:**
  - Notepad++
  - Notepad
  - Not Word or Wordpad
- **On Mac:**
  - **`vi`** or **`vim`**
  - Sublime

blueplanet

Lab: Practice Using Data Structures

- **Complete the following labs:**
  - Lab 1: Create a JSON file
  - Lab 2: Create a JSON Schema
  - Lab 3: Create a HOCON file
  - Lab 4: Create a YAML file

Notify your instructor when you have completed these labs.

blueplanet

Blue Planet Orchestration DevOps Software Skills

Agenda

1	Overview of Software Tools used in Blue Planet
2	Data Structures
3	**Introduction to Devops Toolkit**
4	Python Programming

blueplanet

Virtual Machines

- Provides the ability to run an Operating System within an Operating System
- Used with the Devops environment
- Not typically used on production Blue Planet server
- Most operations handled by Vagrant
- Use Oracle VM Virtual Box Manager to see VMs
- Don’t use Oracle VM Virtual Box Manager to control VMs (start, stop, etc.) if Vagrant is used on system

Vagrant

- **Tool designed to make using VMs easier**
- **VMs are provided in a “box file”: devops-toolkit-orch-18.06.4.box**
- **Steps to use new “box file”:**
  1. Use vagrant init command to initialize: **`vagrant init ra-devops-18.06.4 devops-toolkit-orch-18.06.4.box`**
  2. Edit the Vagrantfile to specify settings:

```
config.vm.network "private_network", ip: "192.168.33.10“
config.vm.provider "virtualbox" do |vb|
        vb.customize ["modifyvm", :id, "--memory", "4096"]
end
config.vm.synced_folder "./shared", "/home/vagrant/shared", type: "nfs"
config.vm.synced_folder ".", "/vagrant", disabled: true
```

  3. Create “shared” directory
  4. Start VM: **`vagrant up`**

blueplanet

Vagrant Commands

- **Key vagrant commands:**
  - **`vargrant up`** – Starts the VM (creates VM first time this is run)
  - **`vagrant halt`** – Shuts down the VM
  - **`vagrant suspend`** – Pauses the VM
  - **`vagrant resume`** (or **`vagrant up`**) – Unpauses a suspended VM
  - **`vagrant destroy`** – Completely removes VM (can be recreated with **`vagrant up`**)
  - **`vagrant ssh`** – Connect to VM via Secure Shell
    - Does not work on Windows natively
    - Use PUTTY when working on Windows

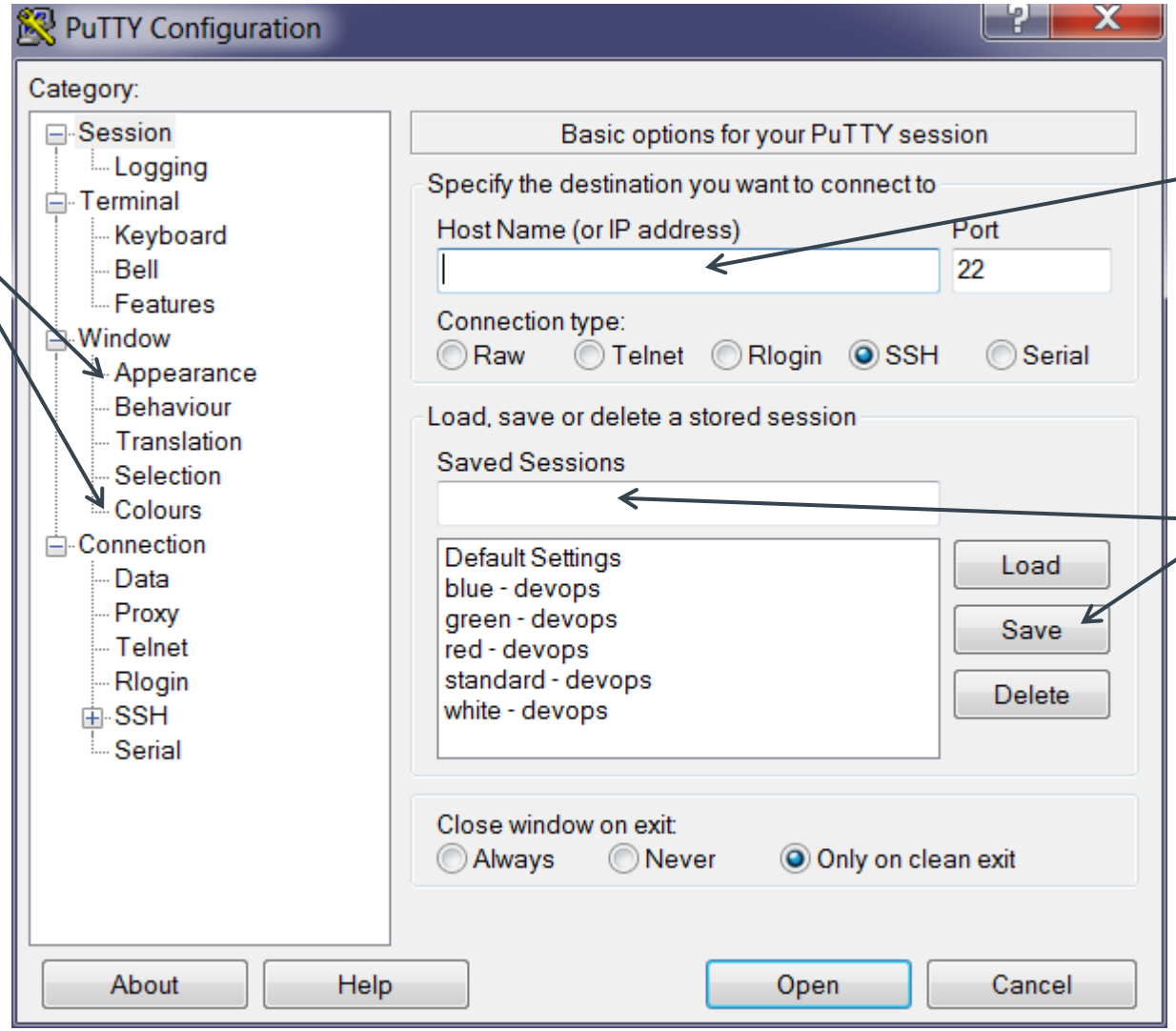
Secure Shell (SSH)

- **Used to connect to VM via a command line login session**
- **Many SSH clients available**
- **PUTTY is recommended**
- **Note: for Devops TK, username and password are both vagrant**

blueplanet

PUTTY

Change SSH features, like font size and colors here



Provide IP address and then click “Open”

Modify features, provide name and then “Save” session

blueplanet

Lab: Configuring the Devops TK

- **Complete the following labs:**
  - Lab 5: Setup the Blue Planet DevOps Toolkit

Notify your instructor when you have completed these labs.

Why docker

- **Docker containers are similar to Virtual Machines with some notable advantages:**
  - Containers use less system resources
  - Containers are quick to start
  - A typical OS can run dozens of containers concurrently
- **Blue Planet uses docker containers to impliment features**
- **These features are called “micro services”**

blueplanet

Docker Basics

- **Developers create docker images**
- **Images contain all of the software needed to create an instance of a docker image**
- **This instance is called a container**
- **The Devops TK includes several images by default, including:**
  - bpocore-dev – The Blue Planet core server
  - frost-orchestrate-ui-dev – Provides a web-based interface to the core server

blueplanet

Docker essentials

- **Essential commands:**
  - **`docker images`** - lists the images that are available on the system
  - **`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

blueplanet

Lab: Controlling docker Containers and the Virtual Machine

- **Complete the following labs:**
  - Lab 6: Starting the Orchestrate Docker Containers
  - Lab 7: Suspending and Restarting the Development Environment

Notify your instructor when you have completed these labs.

blueplanet

Blue Planet Orchestration DevOps Software Skills

Agenda

1	Overview of Software Tools used in Blue Planet
2	Data Structures
3	Introduction to Devops Toolkit
4	**Python Programming**

blueplanet

What is Python?

- **A scripting language that can be used on multiple operating systems, including Linux, Windows and Mac**
- **Some features include:**
  - Easy to create code
  - Robust set of tools (libraries)
  - Highly customizable
  - Great for projects
- **Two primary versions:**
  - 2.x
  - 3.x

blueplanet

What do You Need to Know about Python

- **Depends on your Blue Planet development goal:**
  - To create standard Resource Adapters: Very little experience needed
  - To create customized service templates called imparitive template: at least intermediate level of experience
  - To understand what Blue Planet and Resource Adapter are doing “under the hood”: intermediate-advanced level of experience
- **Focus of this course is just on the basics of Python:**
  - Python configuration files
  - Library basics

Python configuration files

- **Typically contains variable definitions:**
  - String: TYPE_GROUP = 'DockerContainer'
  - String built with function call: RP_CONFIG = os.path.join(RESOURCES_DIR, 'rp_config.yaml')
  - Array: ENDPOINTS = ["cli", "netconf",]
  - Object:

```
STATIC_PATHS = {
        '/images/': os.path.join(os.path.dirname(__file__), 'model/graphics/images'),
            }
```

- **Note: files ending in .pyc are Python compiled code; do not attempt to edit these files**

blueplanet

Python Libraries

- **Blue Planet makes use of many Python libraries**
- **Normally libraries are installed for the entire system**
- **For Resource Adapters, libraries are stored for a specific Resource Adapter using virtualenv**
- **Important to know how to display installed libaries:**
  - System-wide: **`pip list`**
  - Project specifc: ***`project_name`*`/env/bin/pip list`**

blueplanet®

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/**

blueplanet

Building a Resource Adapter Project

- **Full Resource Adapter development is beyond the scope of this class**
- **Building a basic Resource Adapter allows you to make use of tools discussed in later sections**
- **Two steps to creating a basic Resource Adapter:**
  1. The paster command will create all of the template files needed to create a Resource Adapter
  2. The make utility will install all of the software needed to create a Resource Adapter

blueplanet

Render a New RA with rasdk-paste

- **The paster utility**
  - Python utility for generating a new Resource Adapter project from a template
  - Based on PasteScript Python Project
  - Is normally run in a temporary virtual environment
  - An interactive tool that prompts for specifics on how to create the Resource Adapter Project
  - Creates the RA project in the current folder

```
~/shared$ paster
```

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

blueplanet

Understanding make

- **The make utility is used to manage Python software**
- **Uses the Makefile file which is located in the root of the project directory**
- **Common make functions:**
  - **make all** – List all make functions (note: not all enabled in the Toolkit)
  - **make clean** – Remove previously installed Python software

blueplanet

Install Resource Adapter Software with make

- **Simple Makefile for the virtual environment**
  - Includes several **make** targets, including the one to install Resource Adapter Software

```
$ make prepare-venv
```

- **make toolkit-venv**
  - setup.py provides details for the make utilite
  - requirements.txt captures required librarie names and versions
    - Can point to git repos or pypi server
    - All python libraries will be installed

blueplanet

Lab: Resource Adapter Configuration

- **Complete the following labs:**
  - Lab 8: Setting up a Basic Resource Adapter

Notify your instructor when you have completed these labs.

blueplanet

Regular expressions

- **Critical to understand for Resource Adapter development - used in multiple places:**
  - Within TextFSM
  - Matching device prompts
  - Within Jinja templates
- **Used by many languages and tools, including Python, the grep command and the vi editor**
- **Designed to match text using symbols**
- **Resource: http://www.rexegg.com/regex-quickstart.html/**
- **Tester: http://regexr.com/**

blueplanet

Important Regular Expression Characters

- *** = Match zero or more of previous character**
- **+ = Match one or more of previous character**
- **? = Previous character is optional**
- **{*x*,*y*} = Match previous character *x* to *y* times**
- **{x,} = Match previous character *x* or more times**
- **{x} = Match previous character *x* times**

- **^ = Match the beginning of the line**
- **$ = Match the end of the line**

- **\ = Escape the special meaning of a character**

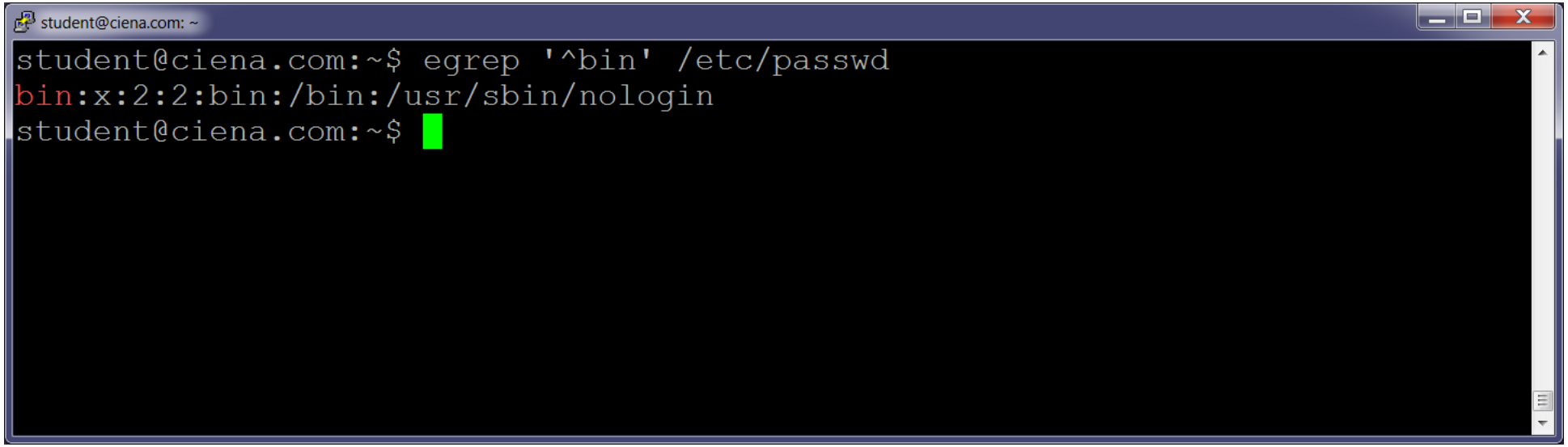
- **. = Match exactly one character**
- **[ ] = Match exactly one character specified within the [ ] characters; examples:**
  - [abc] – Match an a, b or c
  - [0-9] – Match a single digit
- **[^] = Match exactly one character not specified within the [ ] characters; example:**
  - [^abc] – Match any character besides a, b or c

- **| = Or operation**
- **( ) = Group patterns together**

blueplanet

Regular Expression Examples

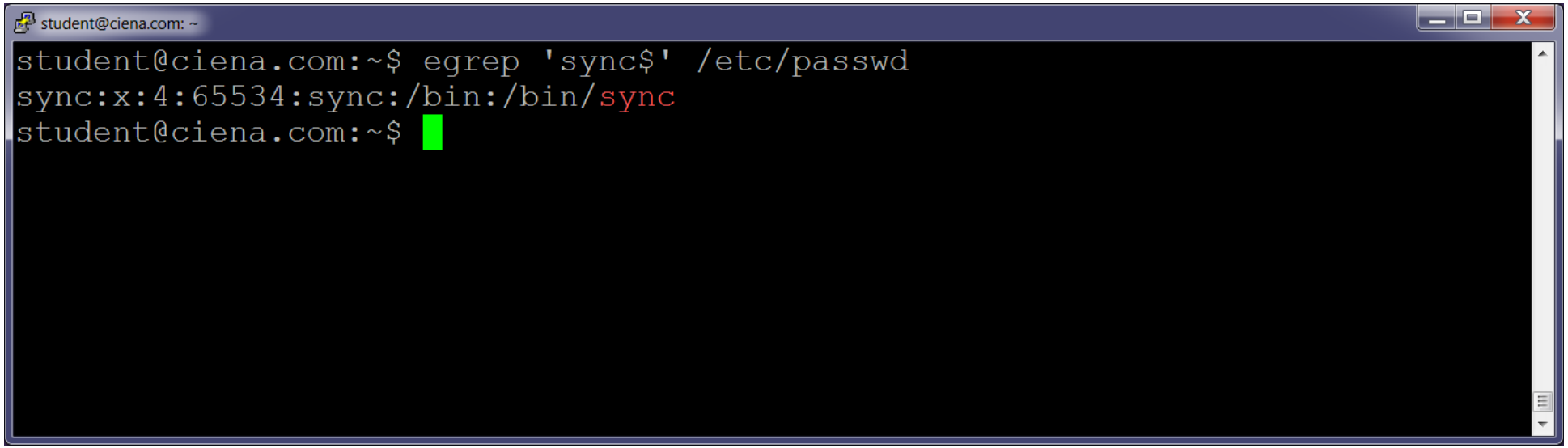
- **The egrep command provides visual examples:**
  - Match a line that begins with "bin"



blueplanet

Regular Expression Examples

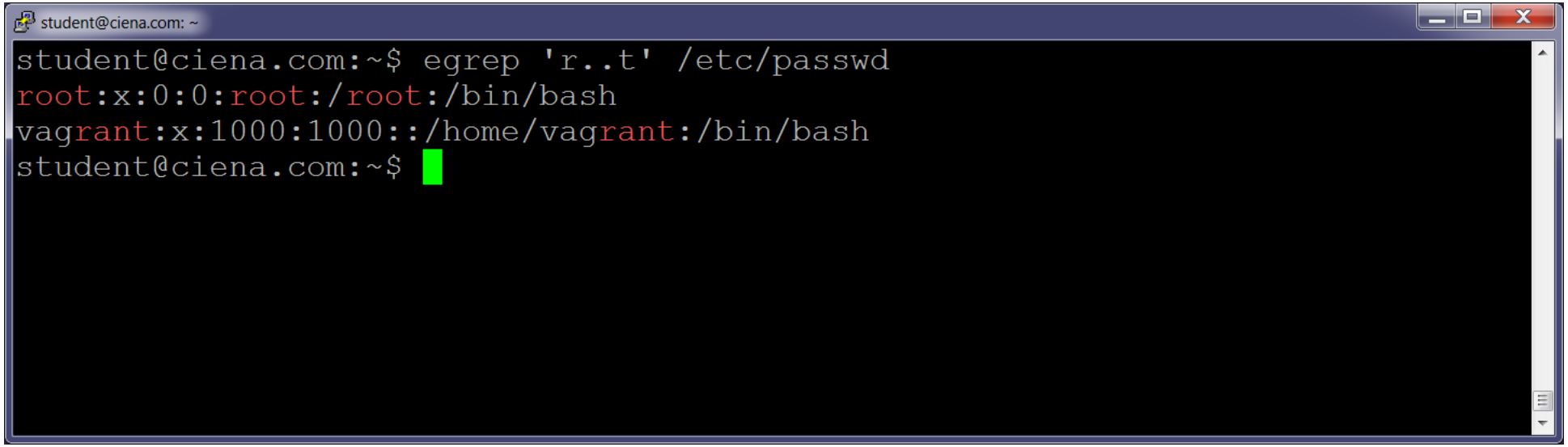
- **The egrep command provides visual examples:**
  - Match a line that begins with "bin“
  - Match a line that ends with “sync“



blueplanet

Regular Expression Examples

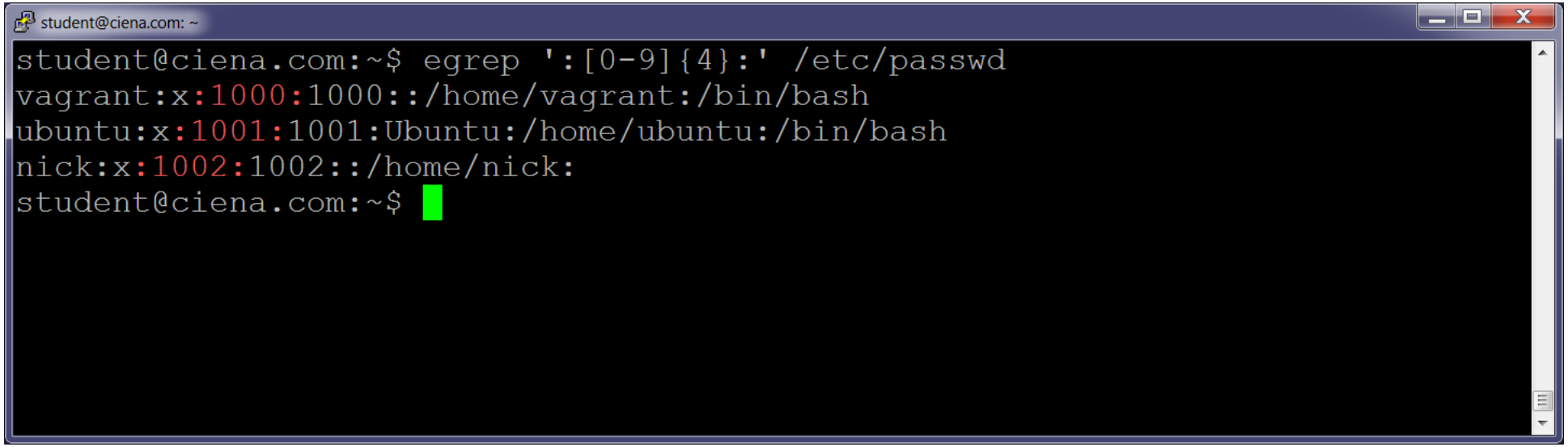
- **The egrep command provides visual examples:**
  - Match a line that begins with "bin“
  - Match a line that ends with “sync“
  - Match a line that contains “r“, any two characters, then “t“



blueplanet

Regular Expression Examples

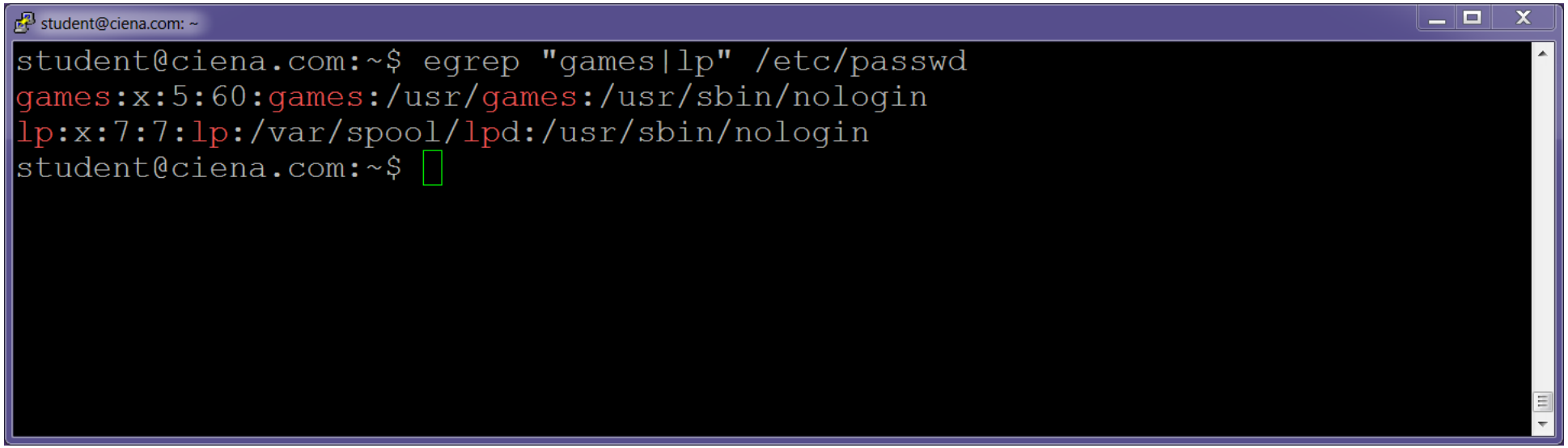
- **The egrep command provides visual examples:**
  - Match a line that begins with "bin“
  - Match a line that ends with “sync“
  - Match a line that contains “r“, any two characters, then “t“
  - Match a line that contains a four digit number between “:“ characters



blueplanet

Regular Expression Examples

- **The egrep command provides visual examples:**
  - Match a line that begins with "bin“
  - Match a line that ends with “sync“
  - Match a line that contains “r“, any two characters, then “t“
  - Match a line that contains a four digit number between “:“ characters
  - Match a line that contains “games” or “lp”



blueplanet

Helpful Suggestions

- **Regular Expressions may behave differently in different tools**
  - The use with Blue Planet is almost always Python-based
  - Exception: In JSON a \ chacter must be represented as \\
- **Regular Expressions are a huge topic**
  - Takes time to learn them
  - Use them with as many tools as possible
  - Practice as much as possible!

blueplanet

TextFSM

- **Used to convert “unformated” data into JSON-structured data**
- **Result is a list of a list (not a full JSON object)**
- **In Blue Planet, TextFSM is used to convert data coming from device to Resource Adapter**
- **TextFSM:**
  - Open source Python package
  - https://code.google.com/archive/p/textfsm/wikis/TextFSMHowto.wiki
  - Implements a template based state machine for parsing semi-formatted text, i.e. CLI response
- **Inputs:**
  - Template file
  - Text input – response from the device

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

Value definitions

State definitions

```
Value Name (\S+)
...
Value Ebs (\d+)

Start
  ^\s+Policy Map ${Name}
  ...
  ^\s+violate-action -> Record
EOF
```


blueplanet

Value and State Definitions

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

    `Value ERPName (regex)`
  - Note: Filldown feature merges line data into a single record (example provided later)
- **State Definitions:**
  - Start newline with **two spaces** followed by `^`
  - End record with `-> Record`
  - Format and spacing must match “record format” of input text
  - Use a combination of text and regex to match format

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

blueplanet

FSM Template Sample

Value definitions

State definitions

```
Value Filldown Name (\S+)
Value Class (\S+)
Value Cir (\d+)
Value Cbs (\d+)
Value Eir (\d+)
Value Ebs (\d+)

Start
  ^\s+Policy Map ${Name}
  ^\s+Class ${Class}
  ^\s+police cir ${Cir} bc ${Cbs} pir ${Eir} be ${Ebs}
  ^\s+conform-action ${Name}
  ^\s+exceed-action${Name}
  ^\s+violate-action ${Name} -> Record

EOF
```

Two spaces at start of each line

One empty line

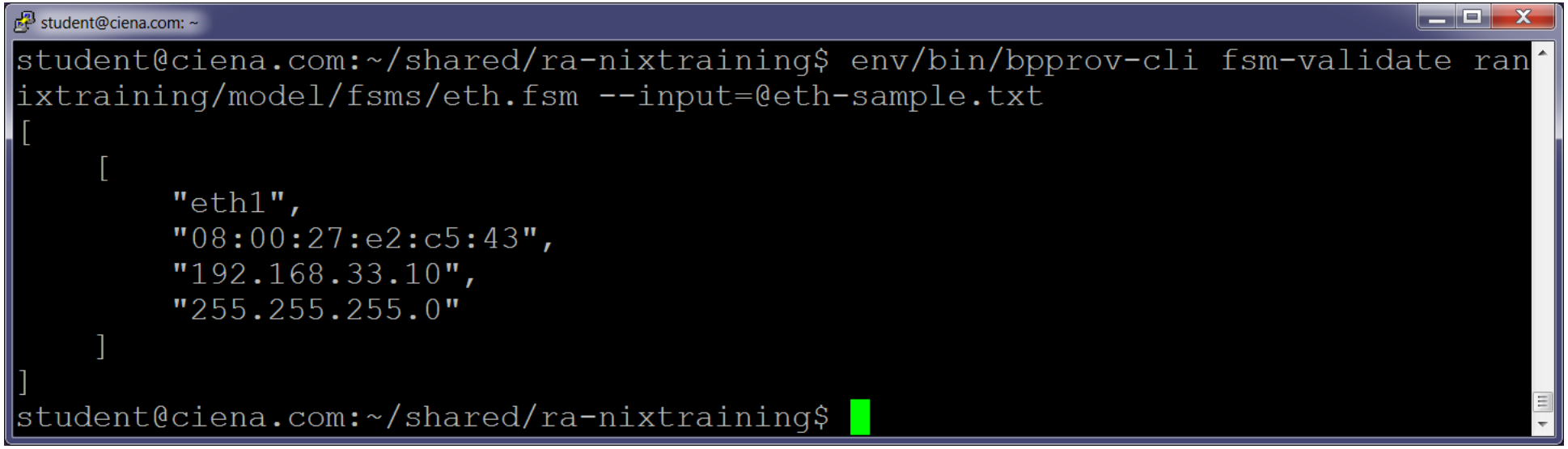
Single space

blueplanet

Using the FSM Testing Tool

- **The Devops Toolkit comes with a FSM testing tool:**
  1. Create the FSM Template and place it in the *project*/model/fsms directory
  2. Copy the output of the device into a plain text file
  3. Execute the following command, replacing *test.fsm* with your FSM Template and *fsm_test.txt* with the file that contains the output of the device:

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



Lab: Parsing Data

- **Complete the following labs:**
  - Lab 9: Use TextFSM to parse data

Notify your instructor when you have completed these labs.

blueplanet

Jinja

- **Used to perform complex data manipulation**
- **In Blue Planet, used by the Resource Adapator to modify data before sending the data to the Blue Planet server**
- **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
- **Live Jinja2 parser: https://cryptic-cliffs-32040.herokuapp.com/**

Jinja example #1

- **Input data:**

```
{ [ {"user": "Steve", "title": "clerk"},
    {"user": "Sue", "title": "manager"} ] }
```

Literally referred to as "data"

- **Jinja template:**

Builds the object structure

```
{ [
  {% for record in data %}
    "name": "{{ record.username|e }}",
    "position": "{{ record.title}}"
  {% endfor %}
] }
```

- **The result:**

```
{ [ {"name": "steve", "position": "clerk"},
    {"name": "sue", "position": "manager"} ] }
```


blueplanet

Jinja example #2

```
{ [
  {% 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 %}
] }
```


blueplanet

Jinja example #3

```
{% set table, name = data.chainId.split(':') %}
sudo iptables --table {{table}} --new-chain {{name}}
{% for rs in data.ruleSpecifications %}
sudo iptables --table {{table}} --append {{name}} {{rs}}
{% endfor %}
```

Lab: Parsing JSON data

- **Complete the following labs:**
  - Lab 10: Use Jinja2 to parse device data

Notify your instructor when you have completed these labs.

blueplanet