

# WISTAR: VIRTUAL NETWORK TOPOLOGY CREATION FOR AUTOMATION SOLUTIONS

Nathan Embery

Senior Consulting Engineer

# LEGAL DISCLAIMER

This statement of direction sets forth Juniper Networks' current intention and is subject to change at any time without notice. No purchases are contingent upon Juniper Networks delivering any feature or functionality depicted in this presentation.

This presentation contains proprietary roadmap information and should not be discussed or shared without a signed non-disclosure agreement (NDA).

# Goals of this presentation

Things you will know once we are all done:

- What is Wistar
- How does it work
- How to use it

# What is Wistar?



BUILD MORE THAN A NETWORK.

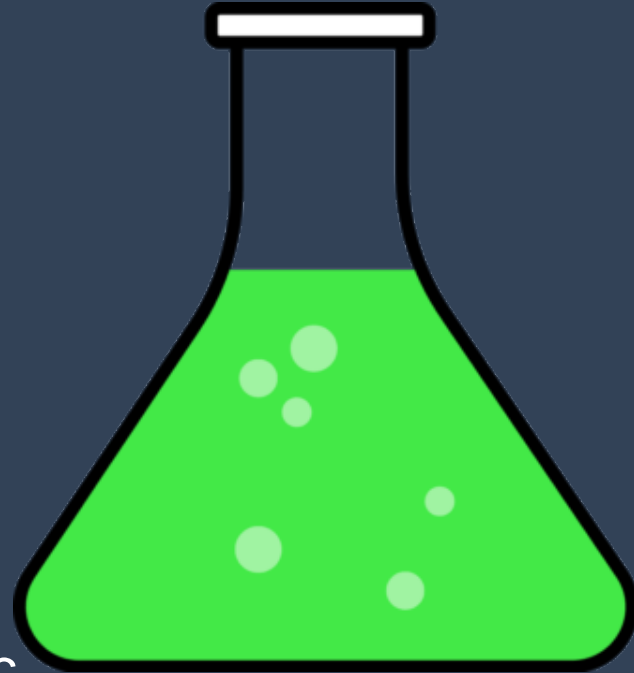
# A tool to harness the power of Virtualization

Virtualization fosters experimentation!

By using virtualization we can:

- launch
- configure
- connect
- save
- suspend

Virtual machines including things like vMX, vSRX, vRR, Contrail, Kubernetes, Openstack, etc



# Building VMs on KVM

To build and connect virtual machines on KVM **requires** the following steps:

- Upload a disk image for each VM
- Define Bridge XML configuration files for each inter-VM connection
- Define a Domain XML configuration file for each VM, including
  - Number of virtual CPUs
  - architecture required (x86\_64)
  - amount of RAM
  - location and type of disk image(s)
  - type of disk controller (ide, scsi)
  - Network interfaces, interface types, mac addresses, and associated bridges
  - Console access (vnc port)

# Example VM definition

```
nembery@faraday:~$ sudo virsh dumpxml t3_vrr02
<domain type='kvm' id='16'>
  <name>t3_vrr02</name>
  <uuid>3de88fe6-b063-4b6c-a591-88b7779eae08</uuid>
  <memory unit='KiB'>786432</memory>
  <currentMemory unit='KiB'>786432</currentMemory>
  <vcpu placement='static'>1</vcpu>
  <resource>
    <partition>/machine</partition>
  </resource>
  <os>
    <type arch='x86_64' machine='pc-i440fx-xenial'>hvm</type>
    <boot dev='hd'>
  </os>
-- snip --
nembery@faraday:~$ sudo virsh dumpxml t3_vrr02 | wc -l 109
```

# Can we automate that?

Wistar **automates** all of this behind some useful abstractions.

Wistar defines several VM image types that **encapsulates** many of the configurable attributes to simplify configuration.

All Domain and Bridge XML files are **templated** and associated with an image type.

The result is that we can **quickly** spin up and experiment on topologies of VMs.



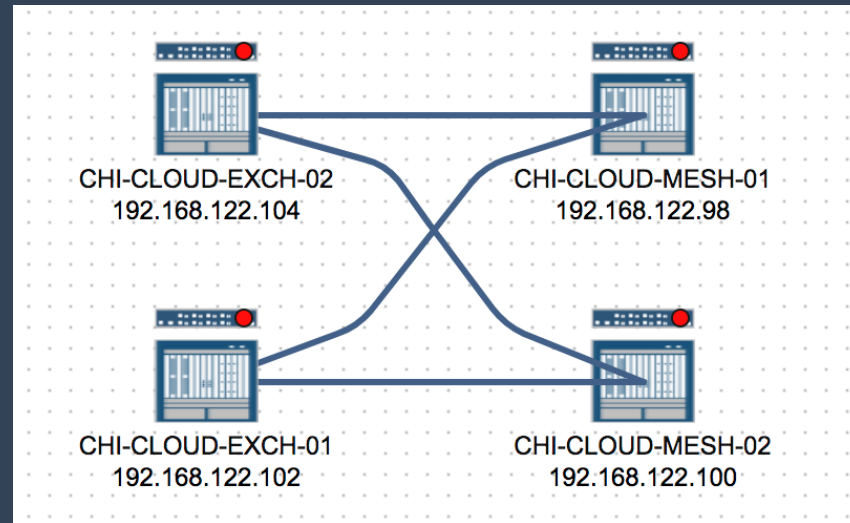
# What is a 'Wistar' anyway?



- From Wikipedia: “The Wistar rat is one of the most commonly used laboratory rats in use today”

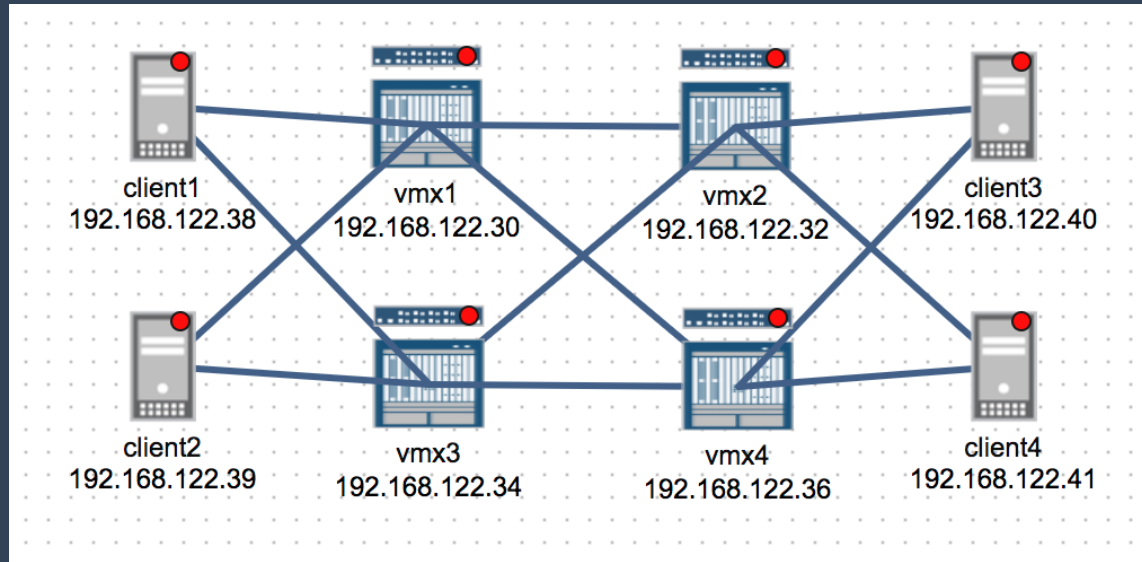
# Simplify building and connecting VMs

Wistar is a tool to simplify creating and connecting virtual appliances in KVM using an intuitive graphical user interface.



# What does it do?

## Step 1: User creates a topology of icons and links them together

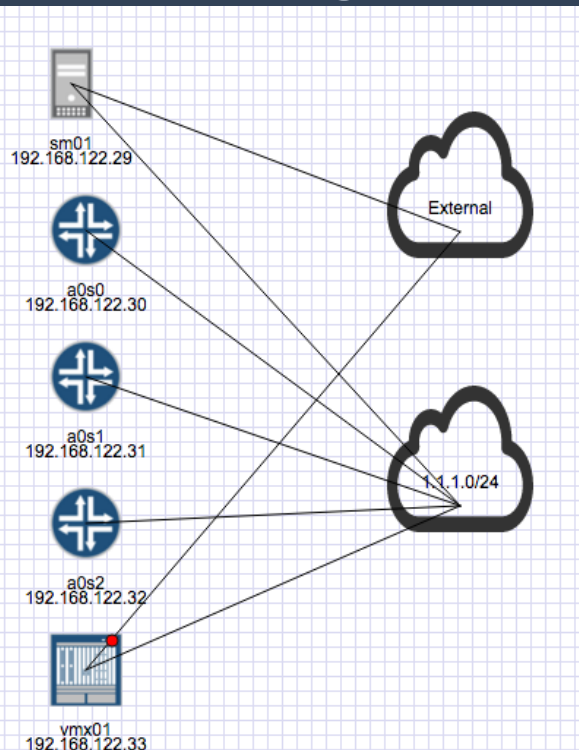


# JSON + Templates

```
{
  "type": "draw2d.shape.node.linux",
  "id": "188a19fb-fa80-4093-ff39-5d00a4b5b795",
  "x": 50,
  "y": 50,
  "width": 30,
  "height": 50,
  "userData": {
    "ip": "192.168.122.42",
    "image": "3",
    "password": "Clouds123",
    "type": "linux",
    "name": "linux03",
    "cpu": "1",
    "ram": "1024",
    "mgmtInterface": "eth1",
    "wistarVm": true,
    "interfacePrefix": "eth",
    "interfaceType": "virtio",
    "configurationFile": "domain.xml",
    "pciSlotOffset": 3,
    "mgmtInterfaceIndex": -1,
    "mgmtInterfacePrefix": "eth",
    "mgmtInterfaceType": "virtio",
    "dummyInterfaceList": [],
    "companionInterfaceList": [],
    "companionInterfaceMirror": false,
    "companionInterfaceMirrorOffset": 19,
    "companionType": "",
    "secondary_disk_type": "",
    "tertiary_disk_type": "",
    "smbios_product_string": "Wistar-linux03-VM",
    "smbios_version": "2.0",
    "smbios_manufacturer": "Wistar",
    "cloud_init_support": true
  },
  "cssClass": "draw2d_shape_node_linux",
  "path": "/static/images/server.png"
},
}
```

Step 2: The icons and connections between them are serialized to JSON and stored in a DB. We can then parse the JSON to determine the **which templates** and **what variables** to use for deployment to the hypervisor.

# Creating Domains and Bridges



## Step 3: Converts JSON into domains and bridges

```
root@feynman:/# virsh list
```

Id	Name	State
10	t16_sm01	running
19	t16_a0s2	running
20	t16_a0s0	running
21	t16_a0s1	running

```
root@feynman:/# virsh net-list
```

Name	State	Autostart	Persistent
default	active	yes	yes
t16_d	active	no	yes
t16_private_br0	active	no	yes

# How does it work?

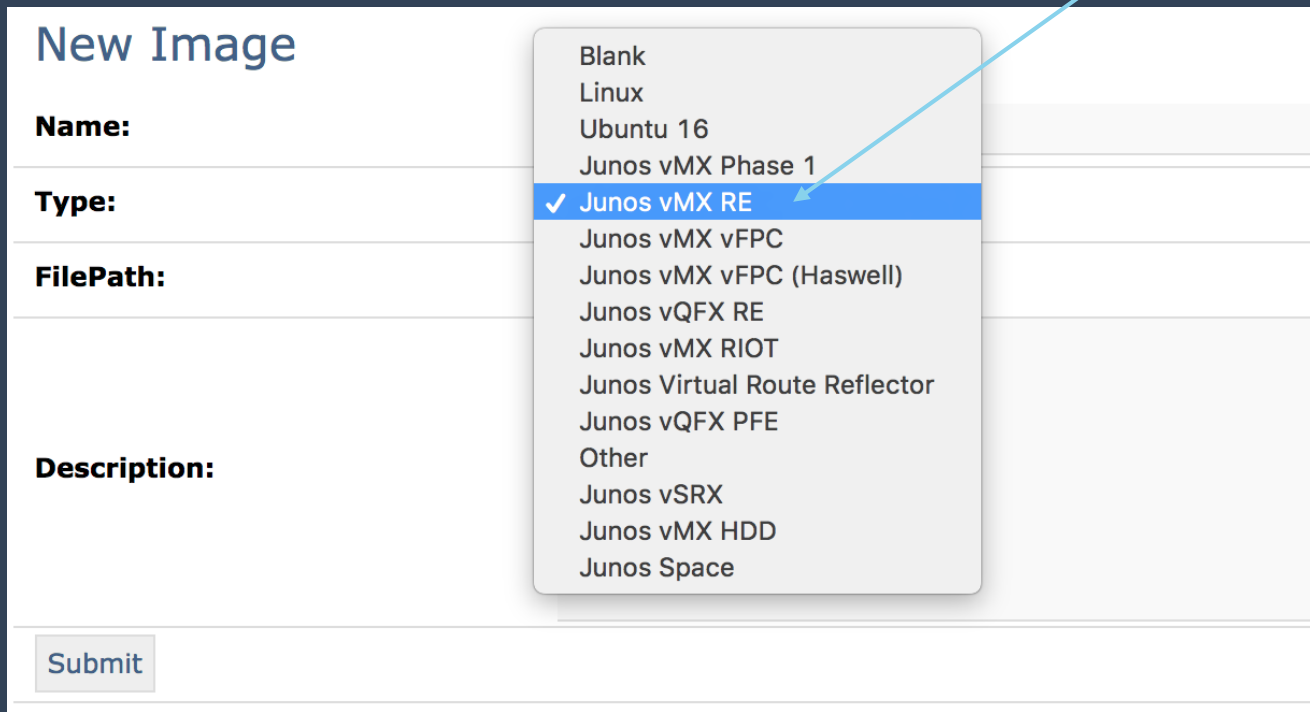
---



BUILD MORE THAN A NETWORK.

# Creating images

The 'Type' attribute is a pre-defined list that is supported by Wistar



The screenshot shows a web form titled "New Image" with the following fields:

- Name:** A text input field.
- Type:** A dropdown menu that is currently open, displaying a list of image types. The option "Junos vMX RE" is selected and highlighted in blue. A blue arrow points from the text above to this option.
- FilePath:** A text input field.
- Description:** A text input field.

At the bottom of the form is a "Submit" button.

The dropdown menu for "Type" contains the following options:

- Blank
- Linux
- Ubuntu 16
- Junos vMX Phase 1
- ✓ Junos vMX RE
- Junos vMX vFPC
- Junos vMX vFPC (Haswell)
- Junos vQFX RE
- Junos vMX RIOT
- Junos Virtual Route Reflector
- Junos vQFX PFE
- Other
- Junos vSRX
- Junos vMX HDD
- Junos Space

# Image Types

Imported disk images become available for use in new topologies

## Imported Disk Images

Name	Description
Ubuntu 14.04	cloud-img
Ubuntu 16.04	cloud-img
vMX 15.1F6.9	vmx 15.1F6.9
vMX 15.1F6.9 Riot PFE	vmx 15.1F6.9 Riot PFE
vRR 16.2R1.6	vrr 16.2R1.6
vSRX 15.1_X49-D60.7	vsrx



# Image Type Details

The 'VM Type' attribute determines many of the defaults that will be used when this VM is deployed to the hypervisor

## Image Detail

Name	vMX 15.1F6.9
Description	vmx 15.1F6.9
VM type	Junos vMX RE
Backing File Path	/opt/wistar/user_images/jinstall64-vmx-15.1F6.9-domestic.img
Provision type	Standalone image
Options	<a href="#">Edit</a> <a href="#">Download</a> <a href="#">Delete</a>

# Adding a VM to a topology

The selected Image type will determine the default values for things like vCPU, vRAM, etc

Add VM

Authentication

root

Clouds123

Instance Name

EMEA-Tech-summit-vmx-01

Base Image

vMX 15.1F6.9

Machine Properties

vCPU: 1

vRAM: 512

MB

Virtual PFE

✓ None

vMX 15.1F6.9 Riot PFE

vPFE vCPU

3

vPFE vRAM

8192

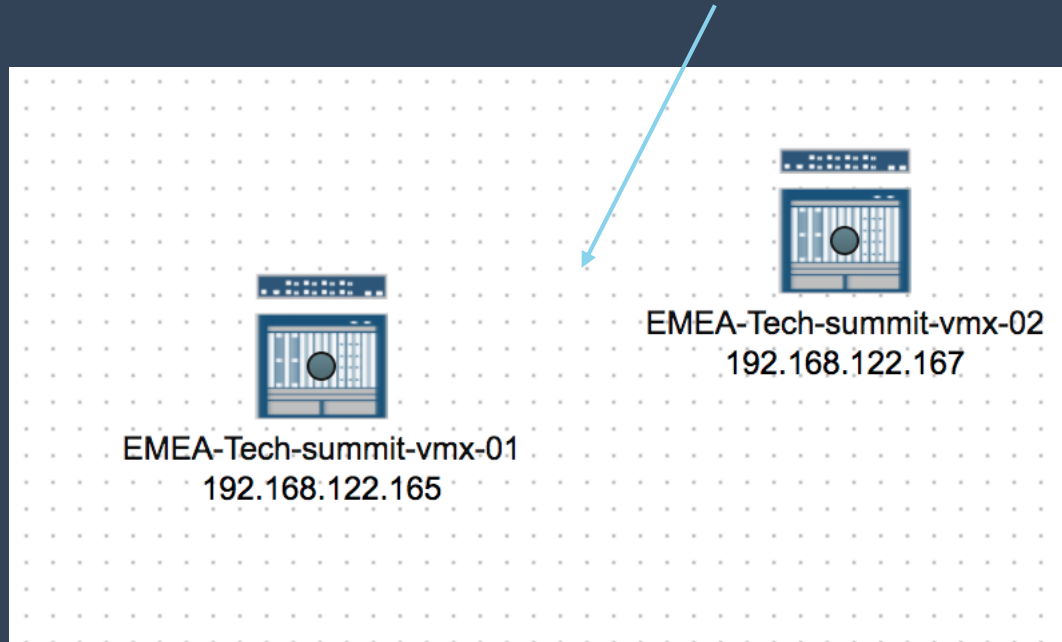
MB

Add and Close

Add Another Instance

# Creating Connections

Connections are created via drag and drop between VM instances.



# Saving a Topology

x

## Save Topology

Name	EMEA-Tech-Summit
Description	2 vmx topology

Save

# Deploying a Topology

Deploying a topology converts the serialized JSON into bridges and domains in the hypervisor

**Topology Details**

Name:

EMEA-Tech-Summit

Description:

2 vmx topology

Edit

Update

Delete

Clone

**KVM Deployment Status**

















Domains not yet defined in hypervisor!

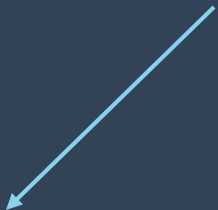
Deploy To Hypervisor

Refresh Hypervisor Status

# Deploying a Topology

Even the simplest topology generates lots of VMs and networks

KVM Deployment Status			
t38_EMEA-Tech-summit-vmx-01	shut off		
t38_EMEA-Tech-summit-vmx-01_child	shut off		
t38_EMEA-Tech-summit-vmx-02	shut off		
t38_EMEA-Tech-summit-vmx-02_child	shut off		
KVM Virtual Network Status			
t38_0_c	running		
t38_2_c	running		
t38_br1	running		
t38_d	running		
<div>Start Topology</div> <div>Pause Topology</div> <div>Refresh Status</div>			




# Device Access

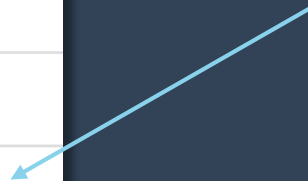


BUILD MORE THAN A NETWORK.

# Console Access

KVM Deployment Status					
t10_SV-M1-VMX01	running				
t10_SV-M1-VMX01_child	running				
t10_SV-M1-VMX02	running				
t10_SV-M1-VMX02_child	running				

Clicking on the Console icon will open a new window





# Management Network

Every VM is automatically connected to an implicit internal management network. By default that network is configured to use the **192.168.122.0/24** subnet.

What interface is connected to the management network is determined by the **image** type configuration.

Wistar uses this network for all automation purposes.

# Integrated Automation

---



BUILD MORE THAN A NETWORK.

# Cloud-Init and Config-Drive

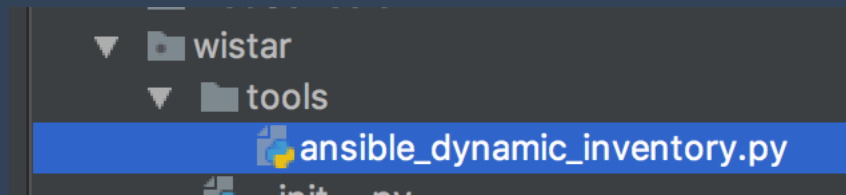
Wistar will use either **cloud-init** or **config-drive** to automatically provision a new VM with the following information:

- User name
- Password
- SSH key
- Management network IP address

# Ansible Dynamic Inventory

Ansible supports a feature called Dynamic Inventory. This allows an external system to supply IP and authentication information to your Ansible environment.

Wistar has a ready to use dynamic inventory script that will consume an exported topology file. Combined with Cloud-init, this enables rapid spin and automation of large complex topologies.



# Junos PyEZ

## Execute cli

show interface terse

Go

Interface	Admin	Link	Proto	Local	Remote
ge-0/0/0	up	up			
lc-0/0/0	up	up			
lc-0/0/0.32769	up	up	vpls		
pfe-0/0/0	up	up			
pfe-0/0/0.16383	up	up	inet		
			inet6		
pfh-0/0/0	up	up			
pfh-0/0/0.16383	up	up	inet		
pfh-0/0/0.16384	up	up	inet		
ge-0/0/1	up	up			
ge-0/0/2	up	up			
ge-0/0/3	up	up			
ge-0/0/4	up	up			
ge-0/0/5	up	up			
ge-0/0/6	up	up			
ge-0/0/7	up	up			
ge-0/0/8	up	up			
ge-0/0/9	up	up			
cbp0	up	up			
demux0	up	up			
dsc	up	up			
em1	up	up			

Built in PyEZ support allows:

- Sync IP information to links
- Execute arbitrary CLI
- Snapshot configurations
- Push configuration templates

# Advanced Topics

---



BUILD MORE THAN A NETWORK.

# Creating new VM Types

All the logic to build a VM is encapsulated in a set of javascript classes.

All VMs inherit from a base class. Because of this, creating new VM types is pretty easy, Just override the attributes you need to customize:

```
draw2d.shape.node.vmx = draw2d.shape.node.wistarStandalone.extend({  
  NAME: "draw2d.shape.node.vmx",  
  INTERFACE_PREFIX: "ge-0/0/",  
  MANAGEMENT_INTERFACE_PREFIX: "em",  
  MANAGEMENT_INTERFACE_INDEX: 0,  
  DOMAIN_CONFIGURATION_FILE: "domain.xml",  
  DUMMY_INTERFACE_LIST: ["1"],  
  ICON_WIDTH: 50,  
  ICON_HEIGHT: 50,  
  ICON_FILE: "/static/images/vmx.png",  
});
```

```
draw2d.shape.node.space = draw2d.shape.node.linux.extend({  
  NAME: "draw2d.shape.node.space",  
  CLOUD_INIT_SUPPORT: false,  
  VCPU: 4,  
  VRAM: 16384,  
  ICON_WIDTH: 50,  
  ICON_HEIGHT: 50,  
  ICON_FILE: "/static/images/space.png",  
});
```

# Multi-VM Icons (vMX phase 3)

One goal was to **simplify** deploying complex multi-vm appliances. Every vm type comes with a set of rules about how it should be **deployed** including things like:

- Which interface is the 'management' interface? fxp0 == em0
- How should companion VMs be connected?
- What type of interfaces are required? virtio, e1000, etc
- What graphical icon should be used?
- How many vCPUs and how much memory is required?
- How are interfaces prefixed? xe-0/0/, ge-0/0/, eth, em
- Etc, etc



# More Information

---



BUILD MORE THAN A NETWORK.

# Where can I get Wistar?

Wistar is open source!

## Open Source Project

- The source code for Wistar can be found at <http://github.com/juniper/wistar>
- All documentation and examples are built and published on [wistar.readthedocs.org](http://wistar.readthedocs.org)
- There is a project to build Wistar using packer at [http://github.com/nembery/wistar\\_packer](http://github.com/nembery/wistar_packer)

# Can our customer's get installation support?

Yes, through the professional services organization and our vLAB solution.

**YES!**

vLAB bring Wistar and OpenStack together for massively large topologies.



# Wistar for VLAB Jumpstart



## Customer Benefits

- Virtualization environment (openstack) with a simulated lab topology using vMX, vRR.
- Automation using **Wistar** for topology creation and NITA for lab spin up and executing automated test cases

## Customer Deliverables

- Installation of Openstack with vMX+vRR Onboarding
- Installation Wistar + NITA
- Knowledge transfer Workshop on vMX, VRR, NITA, Wistar

## Target Problem/Use Case(s)

- Reduce Lab Capex/Opex
- Testing Automation
- Juniper NFV insertion

# VMs that currently work out of the box

- vMX
- vQFX
- vSRX
- vRR
- Junos Space
- Contrail Server Manager
- Contrail Cloud
- Linux Distributions (cloud-init supported by default)

## Links to more information

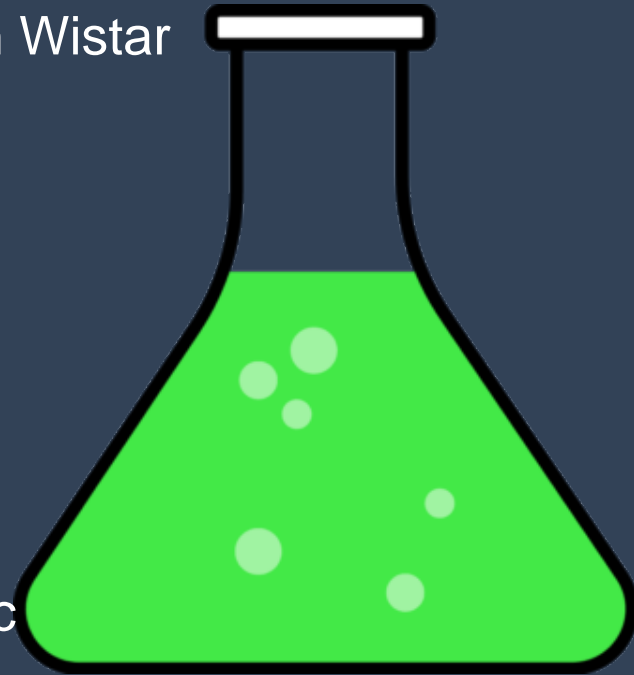
- <https://wistar.readthedocs.io/en/latest/>
- <https://github.com/Juniper/wistar>
- <https://wistar.herokuapp.com/>
- [https://github.com/nembery/wistar\\_packer](https://github.com/nembery/wistar_packer)

# Wistar harnesses the power of Virtualization

Add some examples here of what we can do with Wistar and what we did during the presentation

- launch
- configure
- connect
- save
- suspend

Virtual machines including things like vMX, vSRX, vRR, Contrail, Kubernetes, Openstack, etc



# Thank You!



BUILD MORE THAN A NETWORK.