FRINX

THE OPEN SOURCE NETWORK

FRINX VISION STATEMENT

"Deliver real and sustainable productivity gain by automating processes required to build, operate and grow communication networks."

FRINX TYPICAL USE CASES

- Overall Network Device Automation, including but not limited to
    - LACP link bundles
    - BGP peering services
    - Business internet
    - EVPN
    - L2VPN (VLL, VPLS)
    - L3VPN
    - LLDP topology collection services
    - DOCSIS
    - Amazon VPCs & Direct connect (interface & peering)
- Network Inventory and Change Management
    - Network inventory (heterogeneous platform data transformed to common data model)
    - Operating software management
    - Device configuration
- Workflow management;
    - Network devices
    - Customer care tools
    - Subscriber provisioning tools
    - Billing systems

FRINX STRATEGY

- Develop a controller and agent to connect to customer network devices and provide an upstream network API for our customers.
- Use cloud native software architectures to provide workflow and inventory solutions that control one or many customers network APIs.
- Develop a thriving community to grow our open source device library supporting all device vendors.
- Partner with industry leaders to deliver a multi-tenant, massively scalable cloud based platform for communications and connectivity service providers.

CODE THAT WE WORK ON

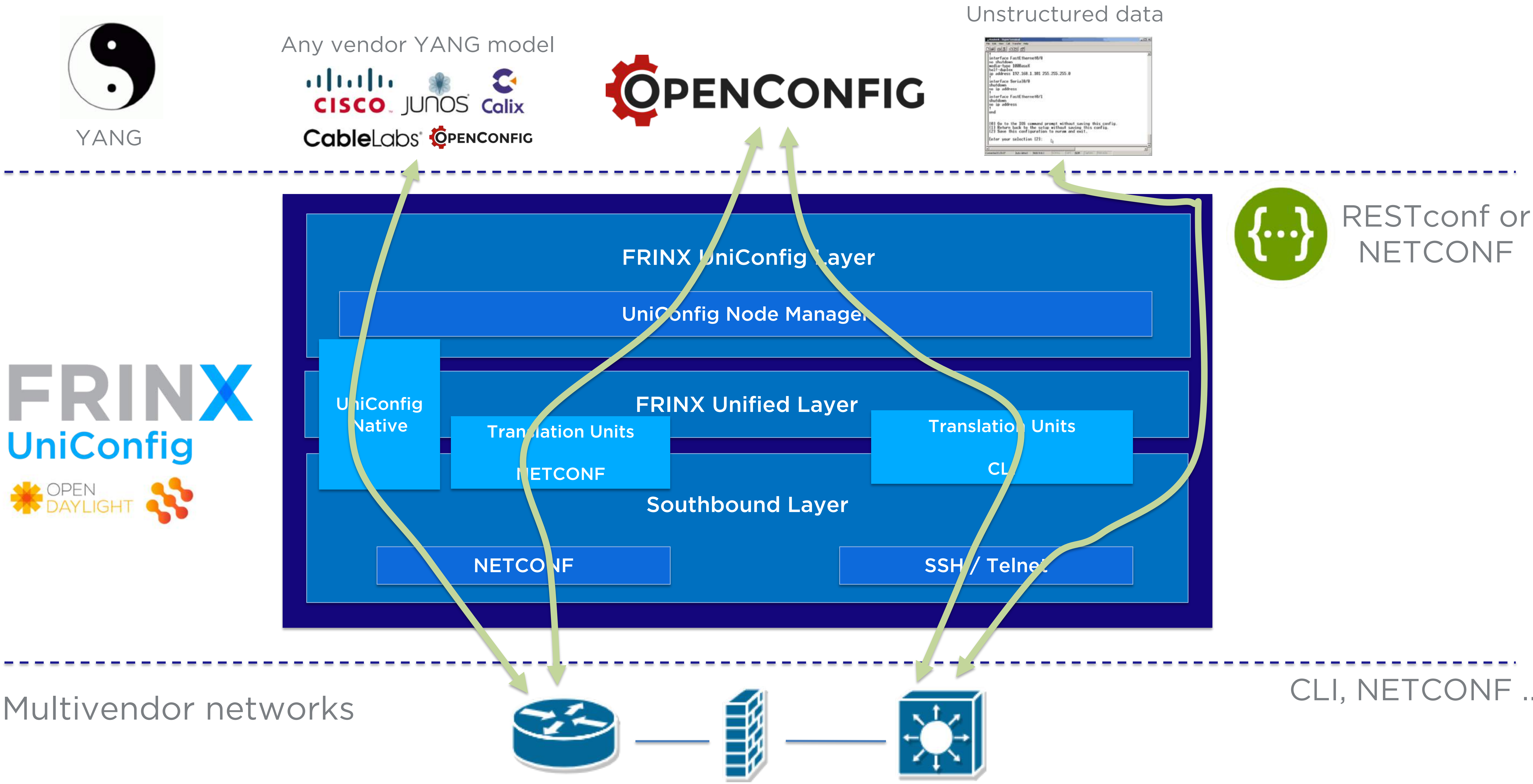
- FRINX UniConfig, with FRINX ODL (for controllers) and Lighty.io (for agents) and deploy with large multi-national customers
- FRINX Open Source device library
- FRINX Machine, a cloud native workflow product based on Netflix Conductor and Elasticsearch
- FRINX contributions to the ODL project (NETCONF, GBP)

FRINX UNICONFIG

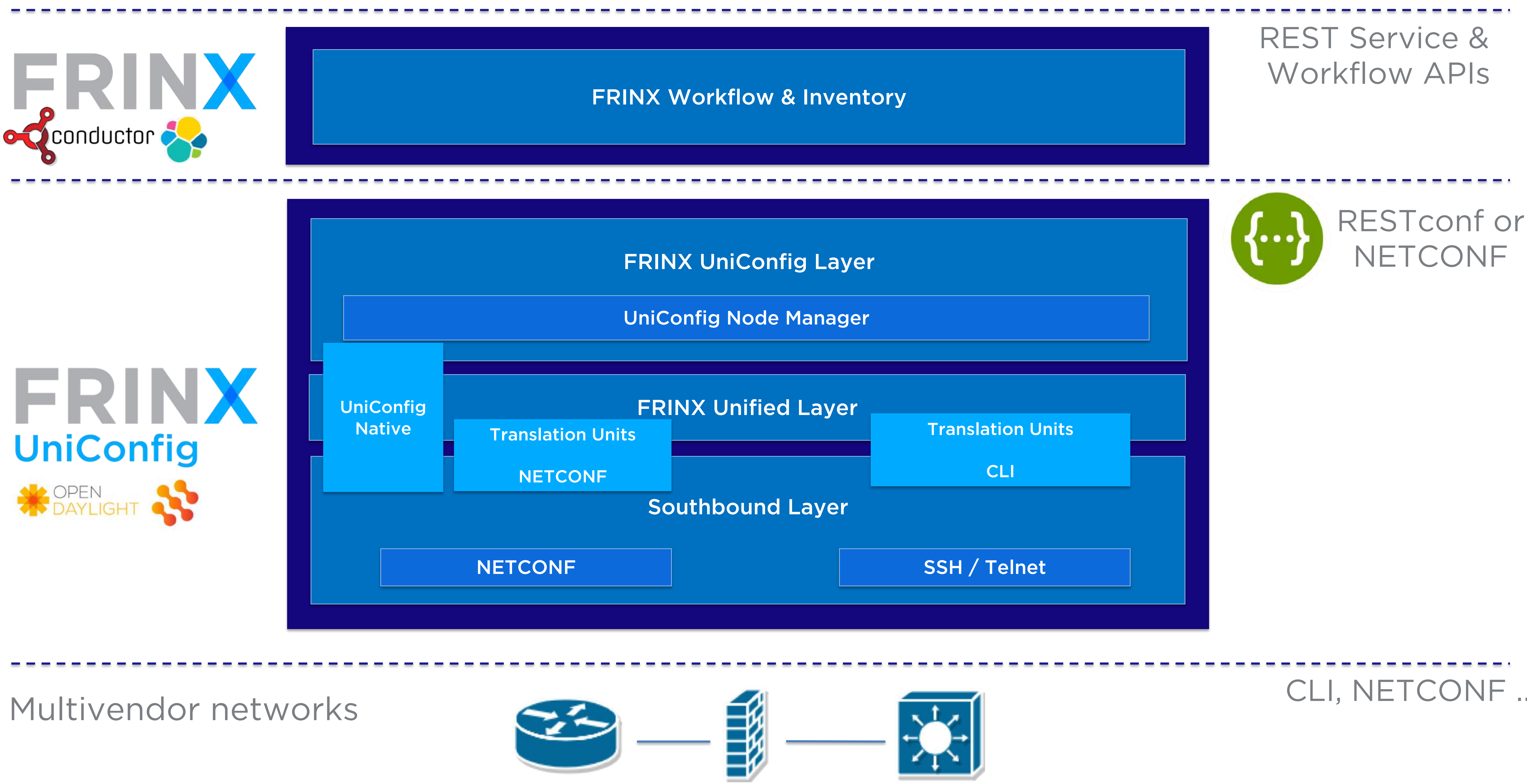
FRINX UNICONFIG ECOSYSTEM

FRINX UNICONFIG NETWORK AUTOMATION

FRINX



FRINX MACHINE - NETWORK AUTOMATION



FRINX UniConfig

Open Source Device Library

FRINX

OSS DEVICE LIBRARY

- Support for stateful translation between CLI (semi-unstructured data) and YANG models
- CLI models are sequence aware. UniConfig service graph is implemented in Create, Read, Update & Delete operations for CLI configs. Required for rollback logic.
- Support for stateful translation between YANG models (e.g. private YANG models translated into OpenConfig and back)
- Stateful means that device configuration is stored in structured format (YANG) in operational and config data stores. This enables UniConfig operations (sync-from-network, diff between config and operational, commit to network, rollback & snapshots) on all device configurations that are mounted in UniConfig.
- Scales up to thousands of devices per controller (with 1000s of lines of config per device)

FRINX UniConfig

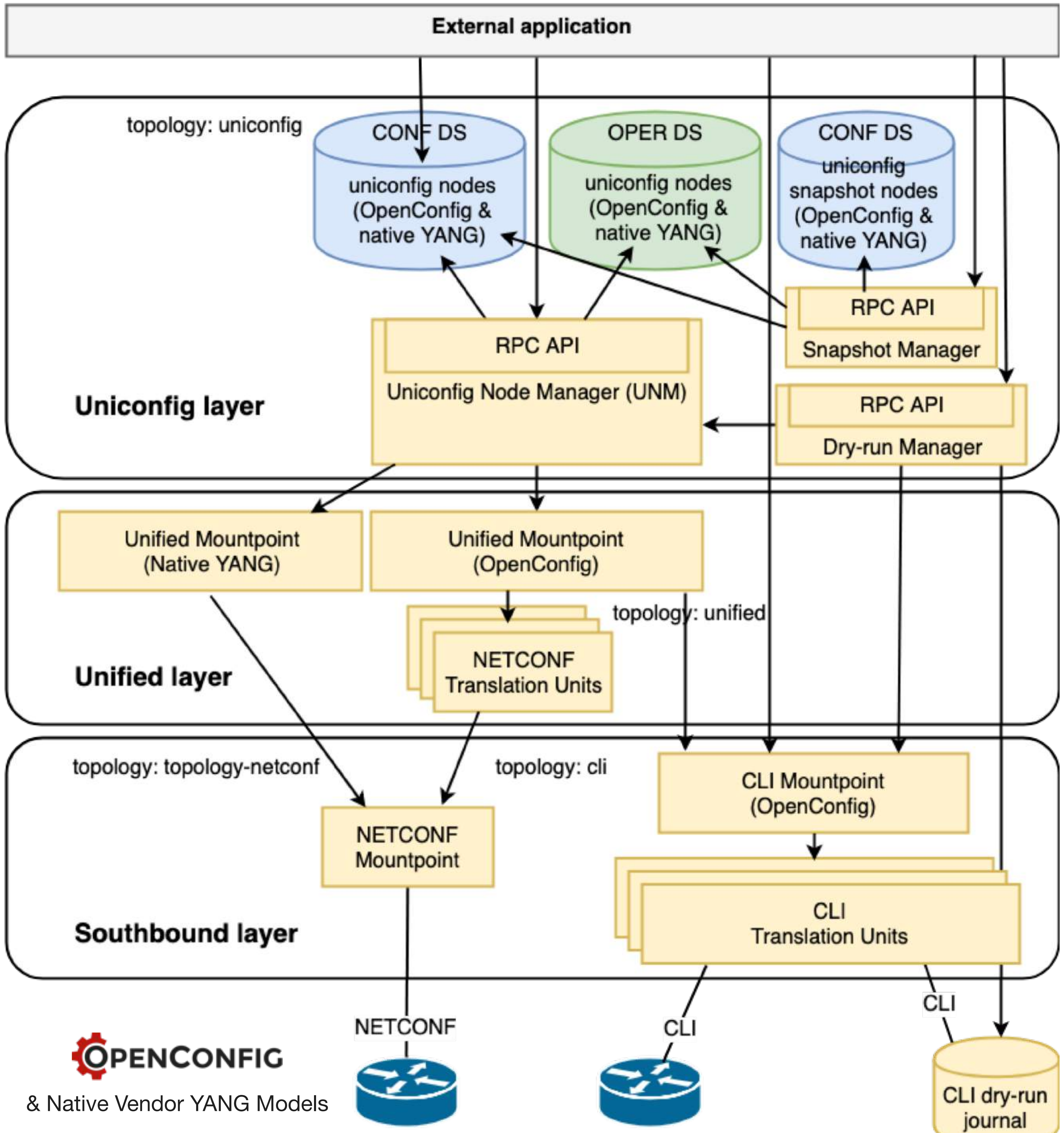
UniConfig Native

UNICONFIG NATIVE

- Read and write to and from devices using their native YANG data models (e.g. Cisco YANG models, JunOS Yang models, CableLabs YANG models …)
- Use the same features on native YANG models as with regular UniConfig OpenConfig models (e.g. sync-from-network, commit, checked-commit, calculate-diff, replace-config-with-operational, rollback, create and load snapshots)
- Works along side UniConfig (some devices can be mounted as UniConfig native, while some devices can be mounted as UniConfig at the same time)
- Available starting with the FRINX ODL 4.2.0 release (April 2019)
- Loading and transformation of YANG models from devices happens on-the-fly. No pre-compilation required.
- Tested scale data point: 1120 devices with 4700 lines of configuration per device, require 3GB of heap

FRINX UniConfig

FRINX SOLUTIONS



FRINX UniConfig Framework – A Layered Architecture

FRINX ODL service components uses layered design where functionality of upper layers depends on the functionality of the layer underneath. Each layer thus provides a higher level of abstraction from the network elements. Applications are allowed to utilize any of the layers in the system.
There are 3 main layers represented by these components (from top to bottom):

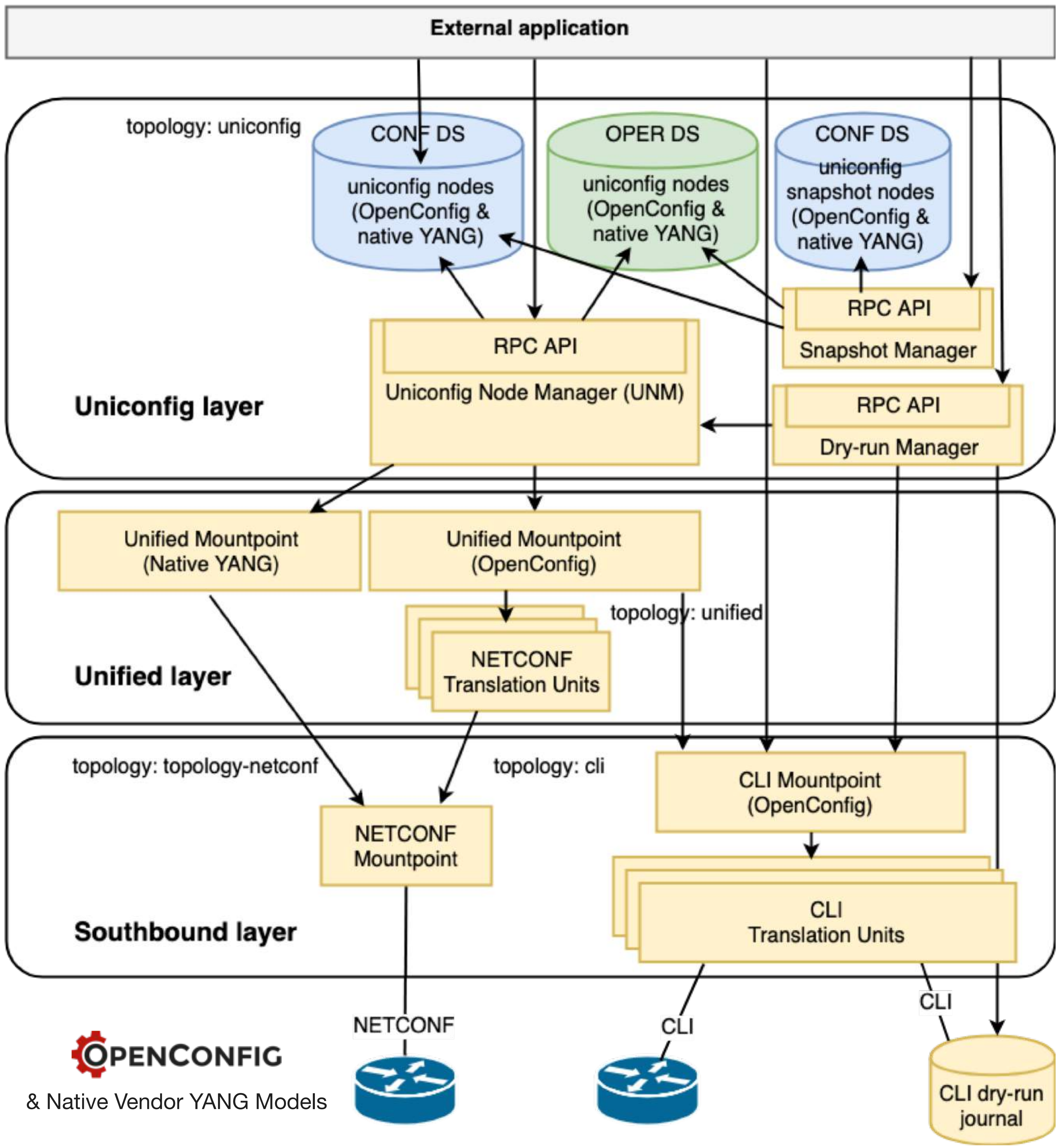
- Uniconfig layer
- Unified layer
- Southbound layer (NETCONF mountpoint, CLI mountpoint with translation units)

The Datastore is a component in ODL which stores structured data described by YANG models. There are two separate Datastores:

- Config datastore (CONF DS) - contains intended state (intended device configuration). This datastore is persistent and external (outside ODL) applications have read/write access.
- Operational datastore (OPER DS) - contains actual state (actual device configuration). OPER DS is not persistent and external applications have read only access.

Mountpoints in ODL represent communication interfaces with an external system. Mountpoints are usually registered under a node in a topology.

FRINX UNICONFIG & UNICONFIG NATIVE



Business application using UniConfig

Device abstraction provides API to create, read, update and delete device configurations in common OpenConfig Format. Config data store contains intended configurations of all network devices, while operational data store contains all current configurations. Network transaction capabilities provide commit, snapshot and rollback functions across one or multiple devices.

Unified layer provides the ability to combine devices connected with different transport protocols and different models into one common representation. Includes open source device library (YANG <-> YANG) and the ability to interact with vendor YANG models (UniConfig native)

Southbound layer provides connectivity to devices via multiple protocols (NETCONF, SSH, Telnet, …). Includes open source device library (YANG <-> CLI)

FRINX

OPERATIONS

- UniConfig Manager
  - sync-from-network
  - commit
  - checked-commit
  - calculate-diff
  - replace-config-with-operational

- Dry-run Manager
  - dry-run – works only with CLI nodes

- Snapshot Manager
  - create-snapshot
  - delete-snapshot
  - replace-config-with-snapshot

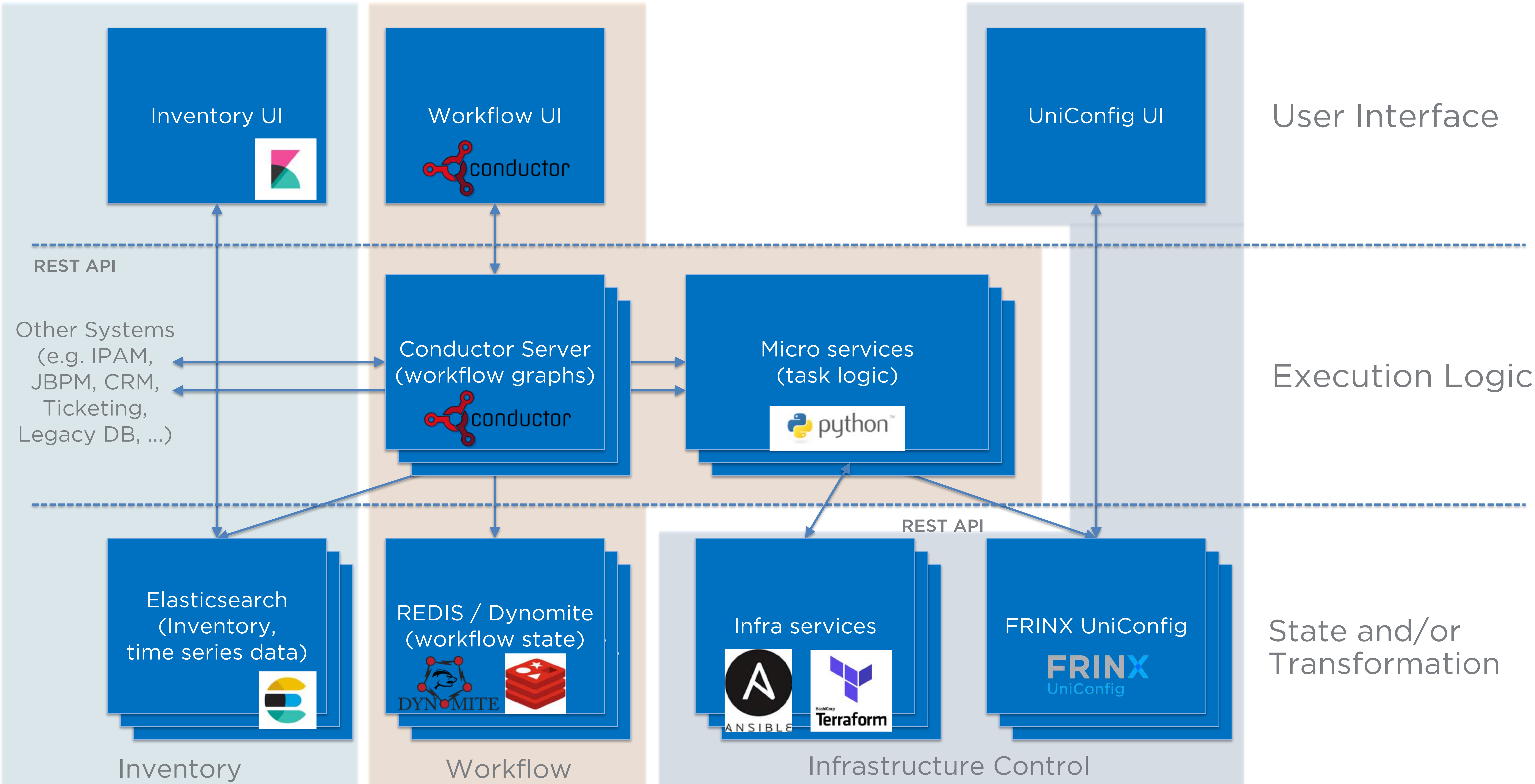
FRINX machine

FRINX MACHINE - NETWORK AUTOMATION

FRINX

FRINX MACHINE - NETWORK AUTOMATION
FRINX
FRINX
conductor
Workflow & Inventory
REST Service &
Workflow APIs
FRINX UniConfig Layer
UniConfig Node Manager
FRINX
UniConfig
Translation Units
CLI
Southbound Layer
NETCONF
SSH / Telnet
python
HashiCorp
Terraform
ANSIBLE
RESTconf or
NETCONF
μ-services
FRINX
UniConfig
OPEN
DAYLIGHT
Stateful
Stateless
Multivendor networks
CLI, NETCONF ...

FRINX MACHINE – COMPONENTS



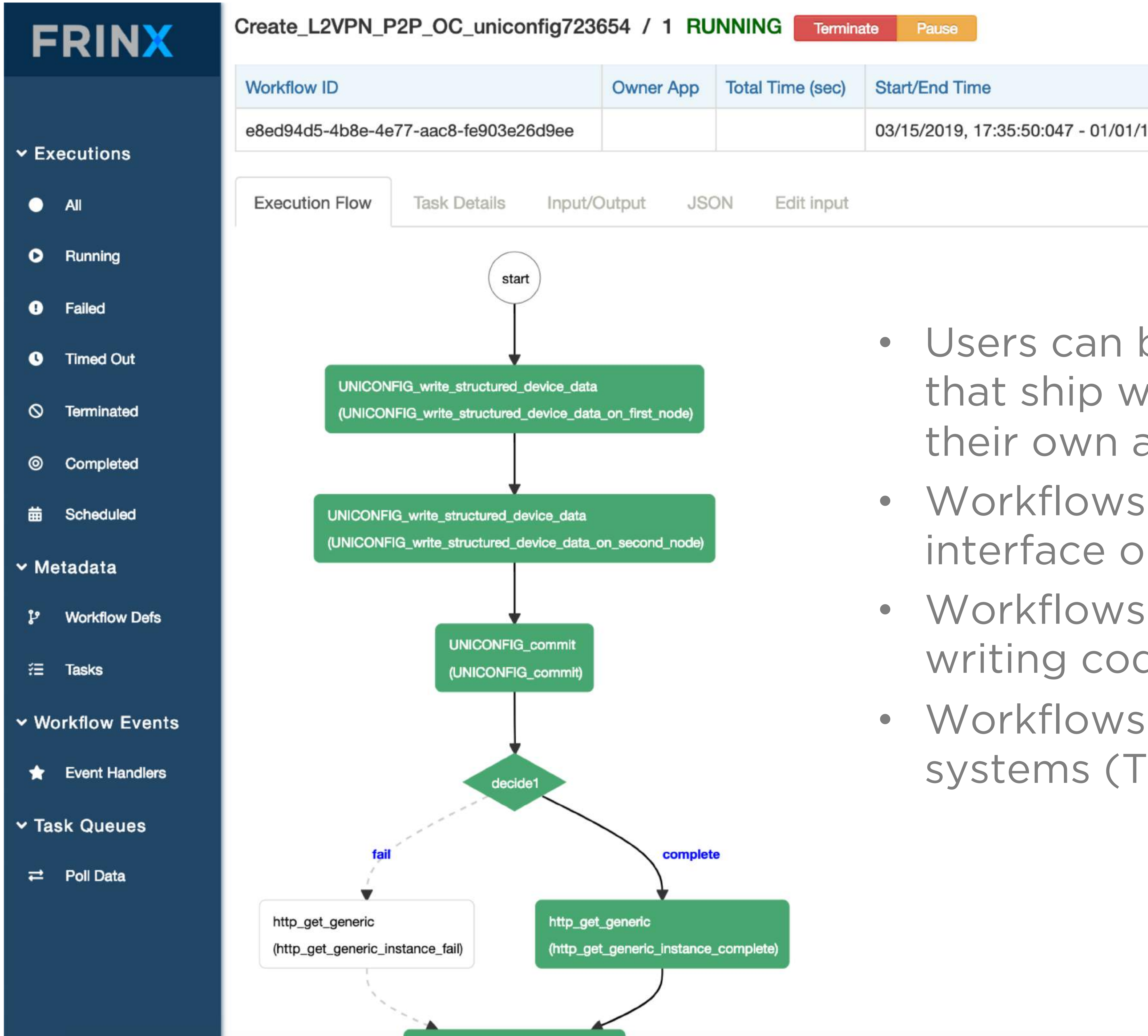
Key Solution Tenets

- Use existing solutions wherever available (OpenDaylight, Elasticsearch, Conductor, Ansible, Terraform, …)
- Provide stateful and stateless interaction with network infrastructure
- Provide a framework for components and how they interact
- Provide out-of-the-box workflows & services
- Provide open source device library
- Provide customer access to all source code
- Provide solution support for enterprises and operators
- FRINX Machine fits in 6 GB RAM / 30GB disk and installs and starts in a few minutes

FRINX MACHINE

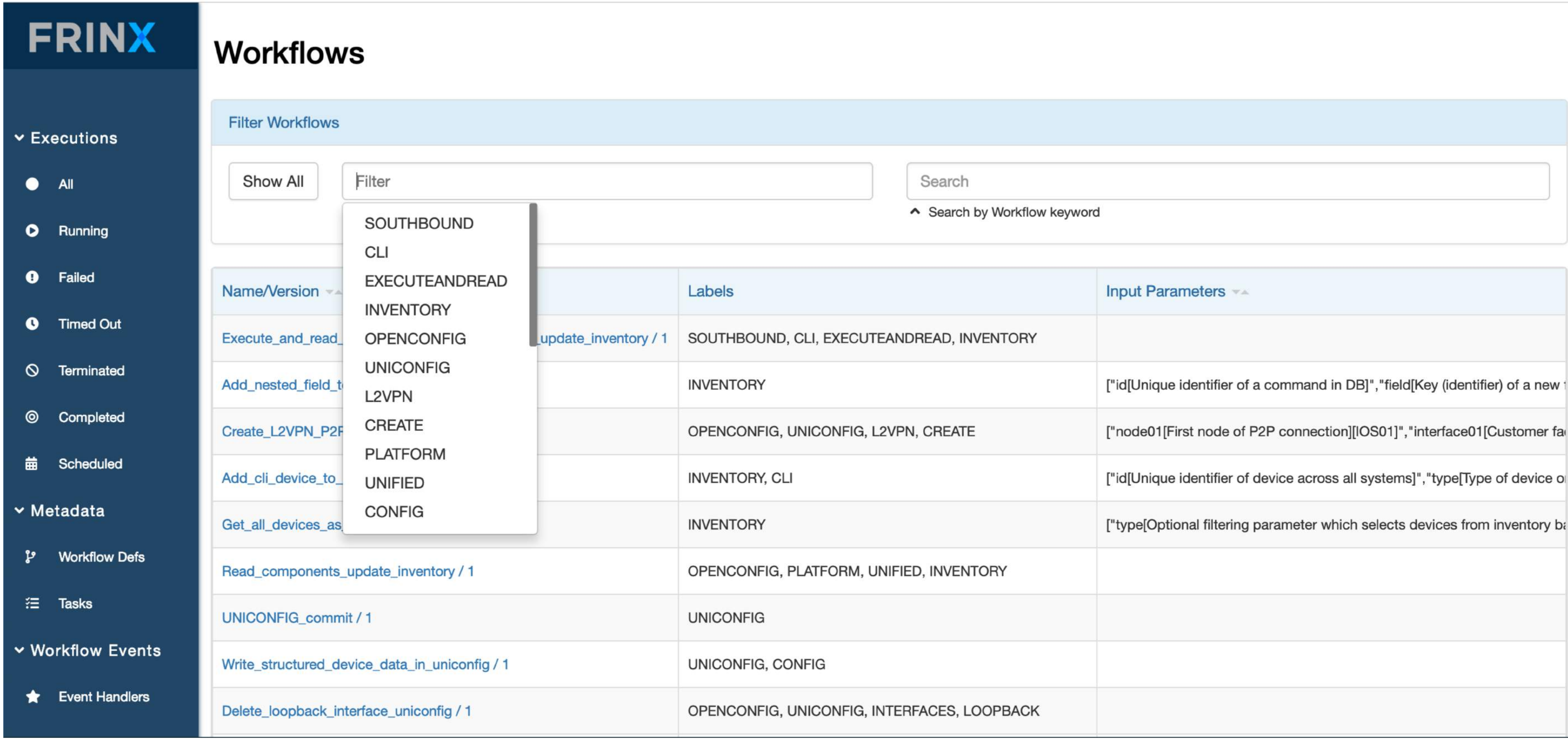
User Interface

FRINX Workflow UI

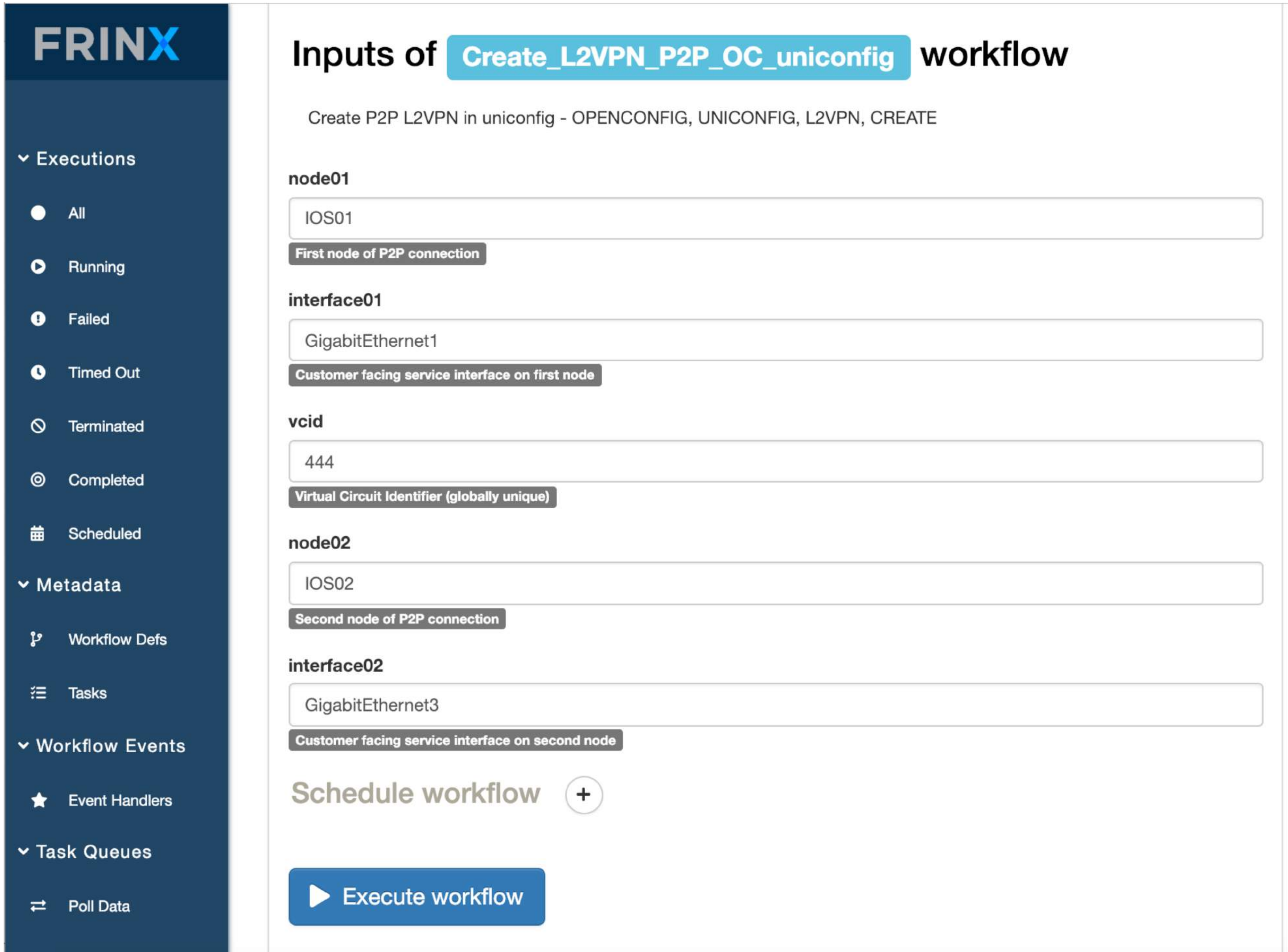


- Users can build on a library of workflows that ship with FRINX Machine to create their own automation workflows.
- Workflows can be started via REST interface or GUI.
- Workflows can be created without writing code
- Workflows can integrate with other systems (Ticketing, E-mail, Slack, …)

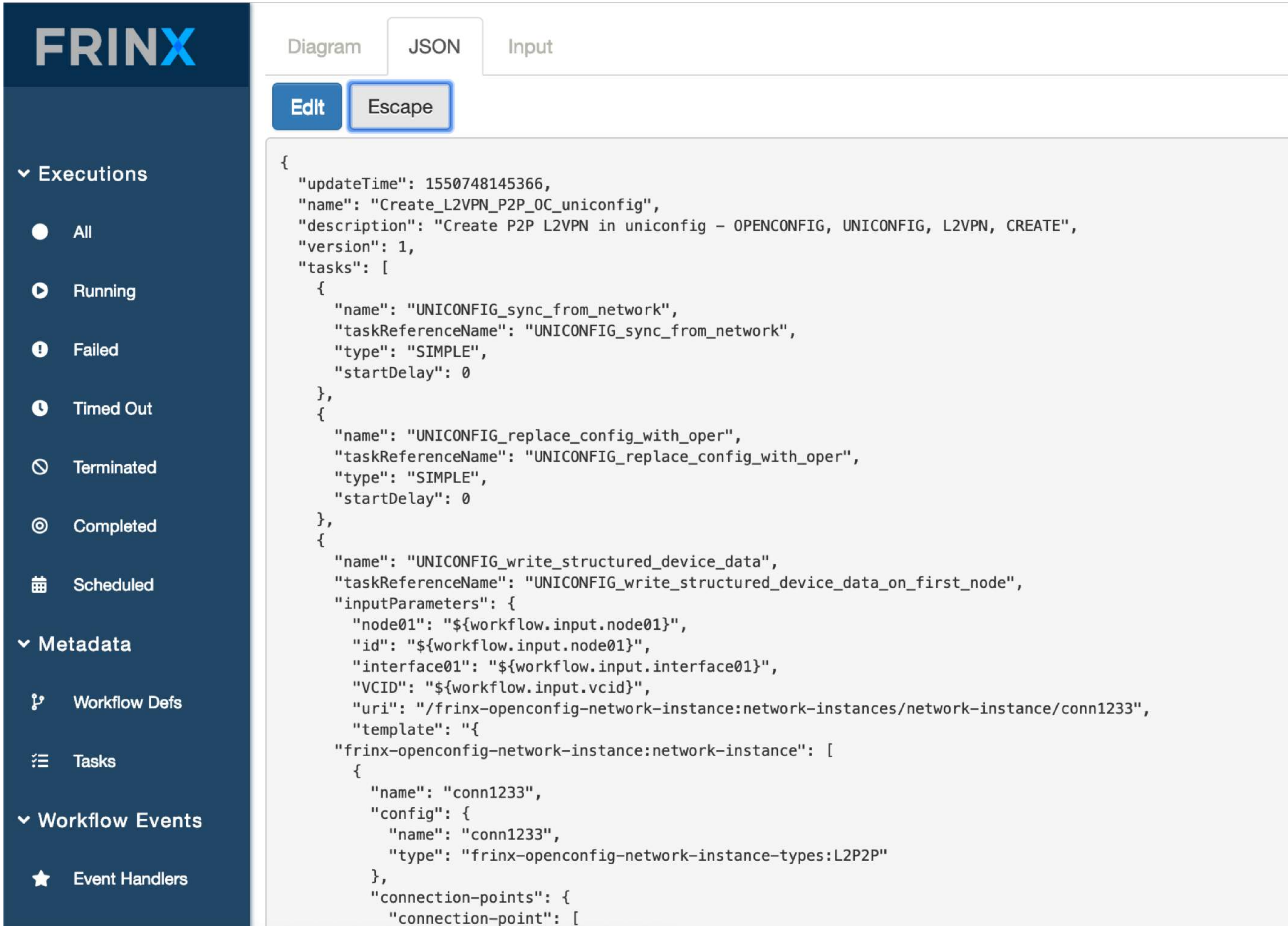
FRINX Workflow UI



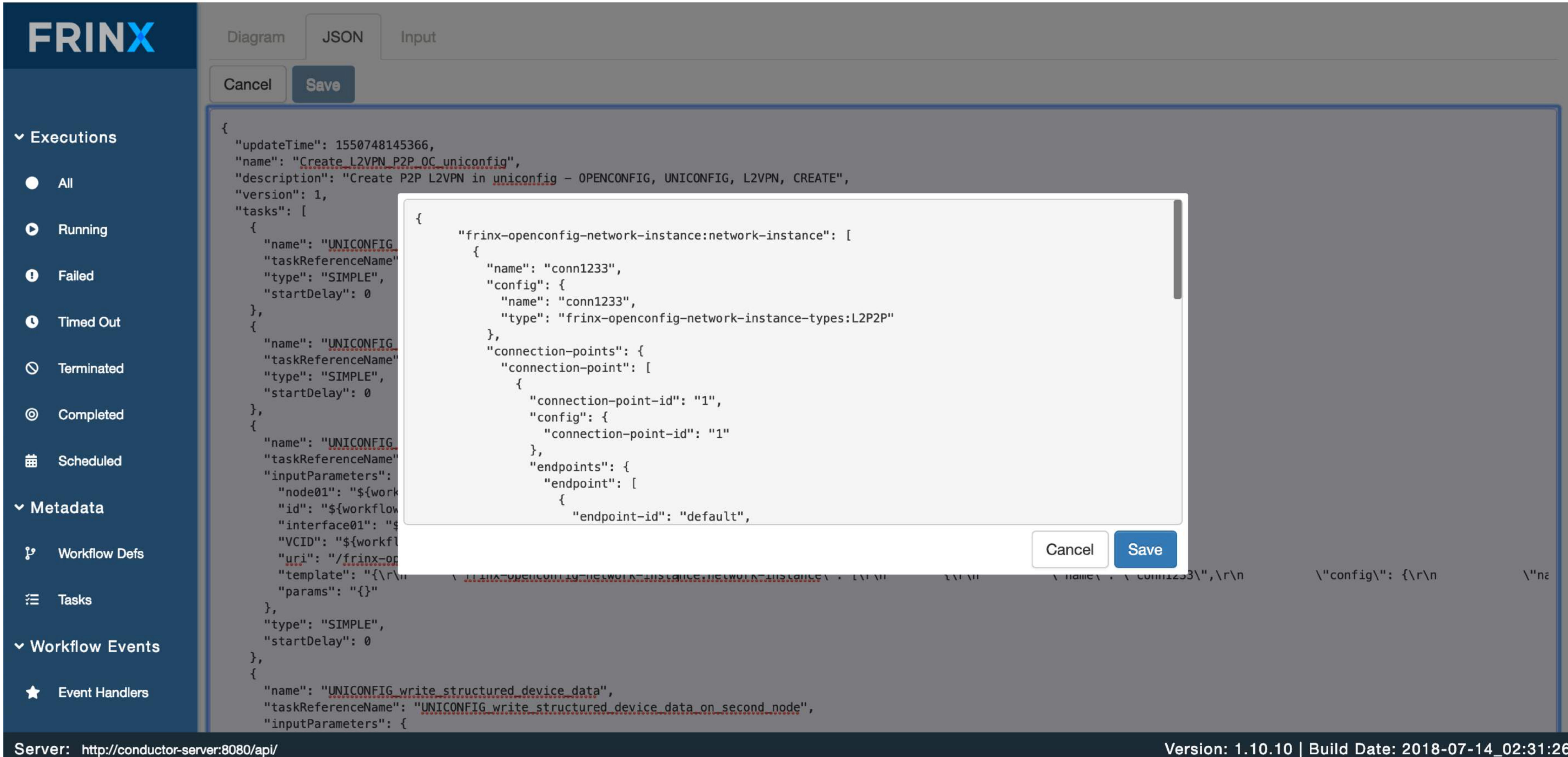
FRINX Workflow UI



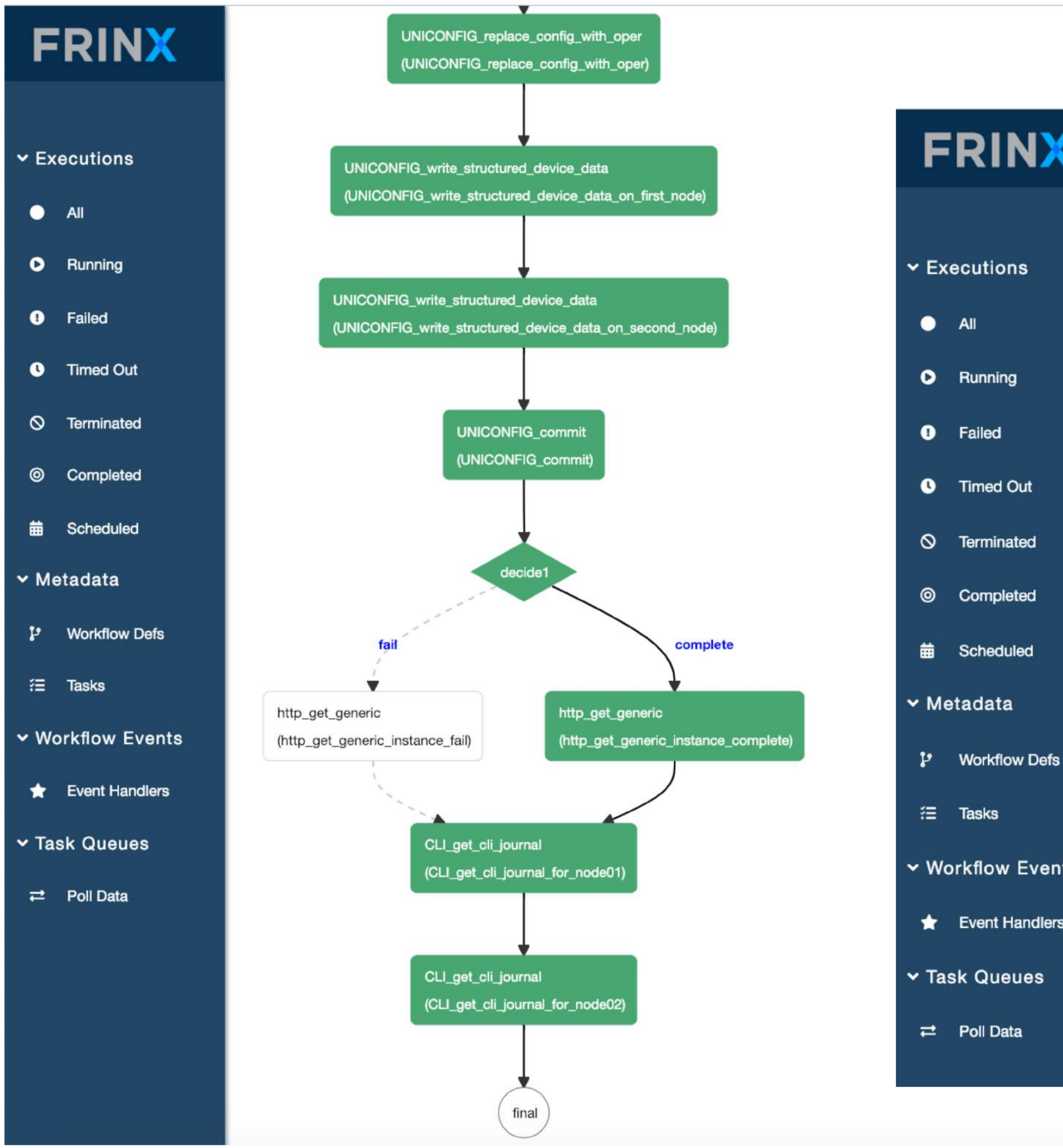
FRINX Machine Workflows – Edit Workflow

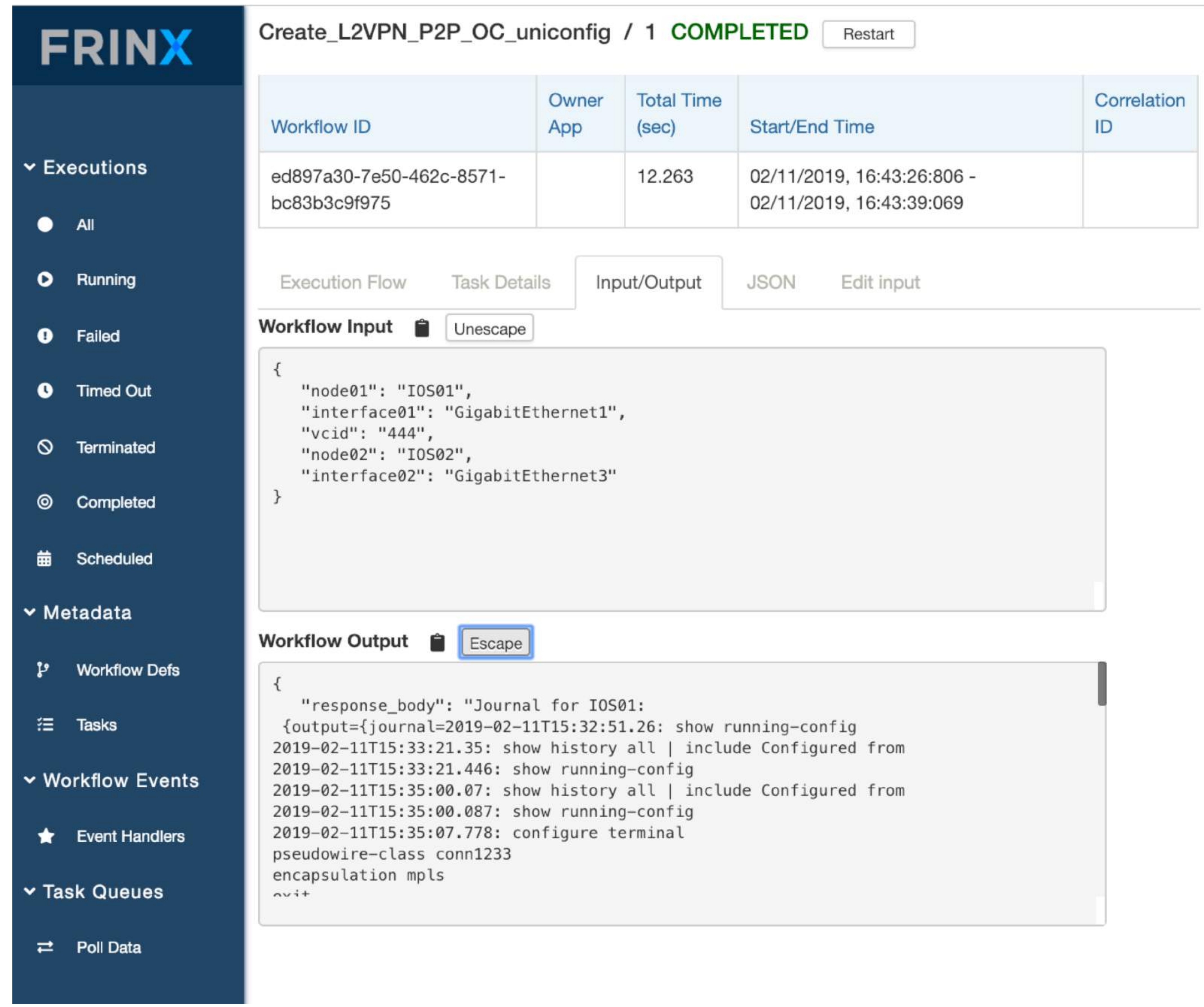


FRINX Machine Workflows – Edit Workflow

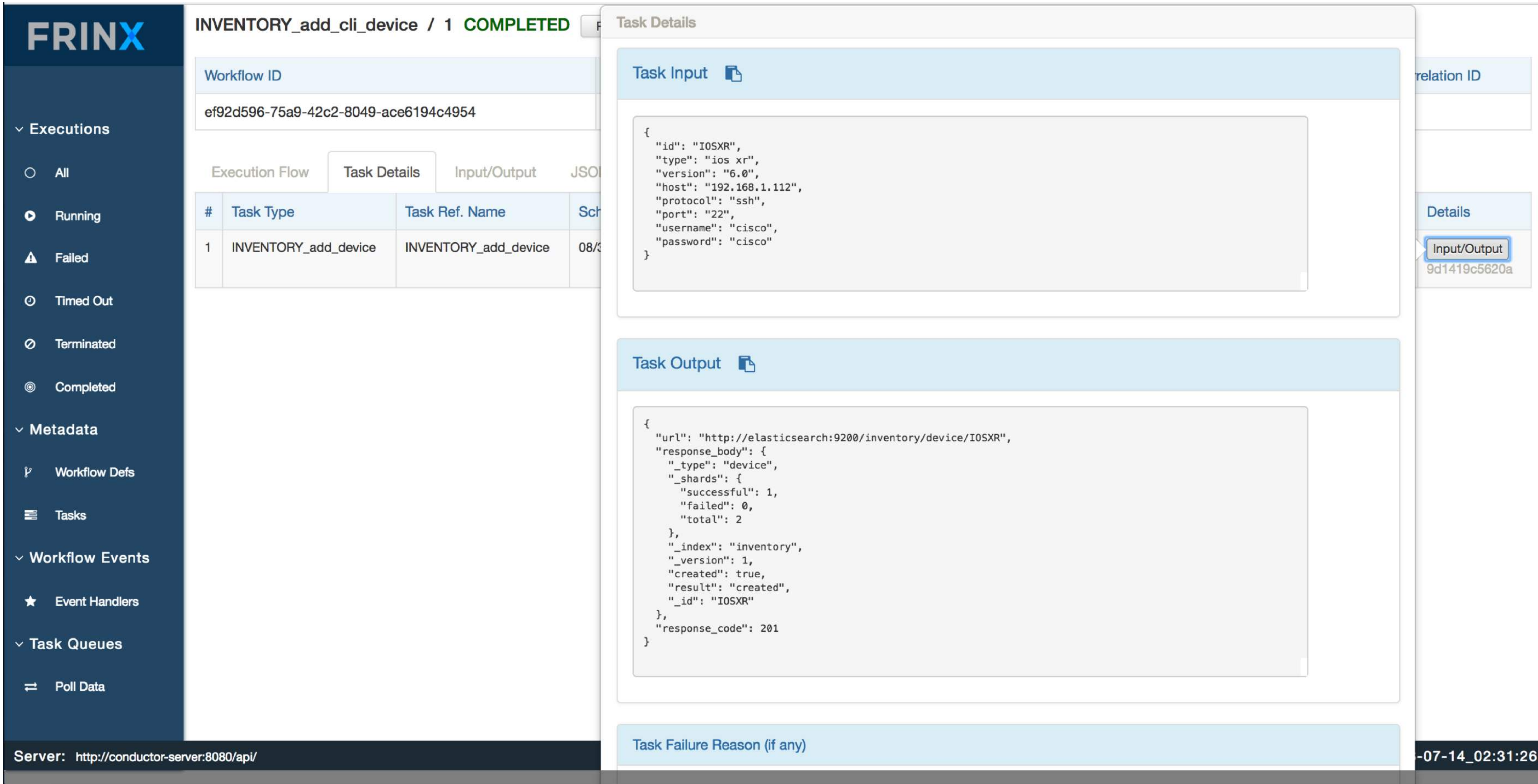


FRINX Machine Workflows – Detailed Execution Information

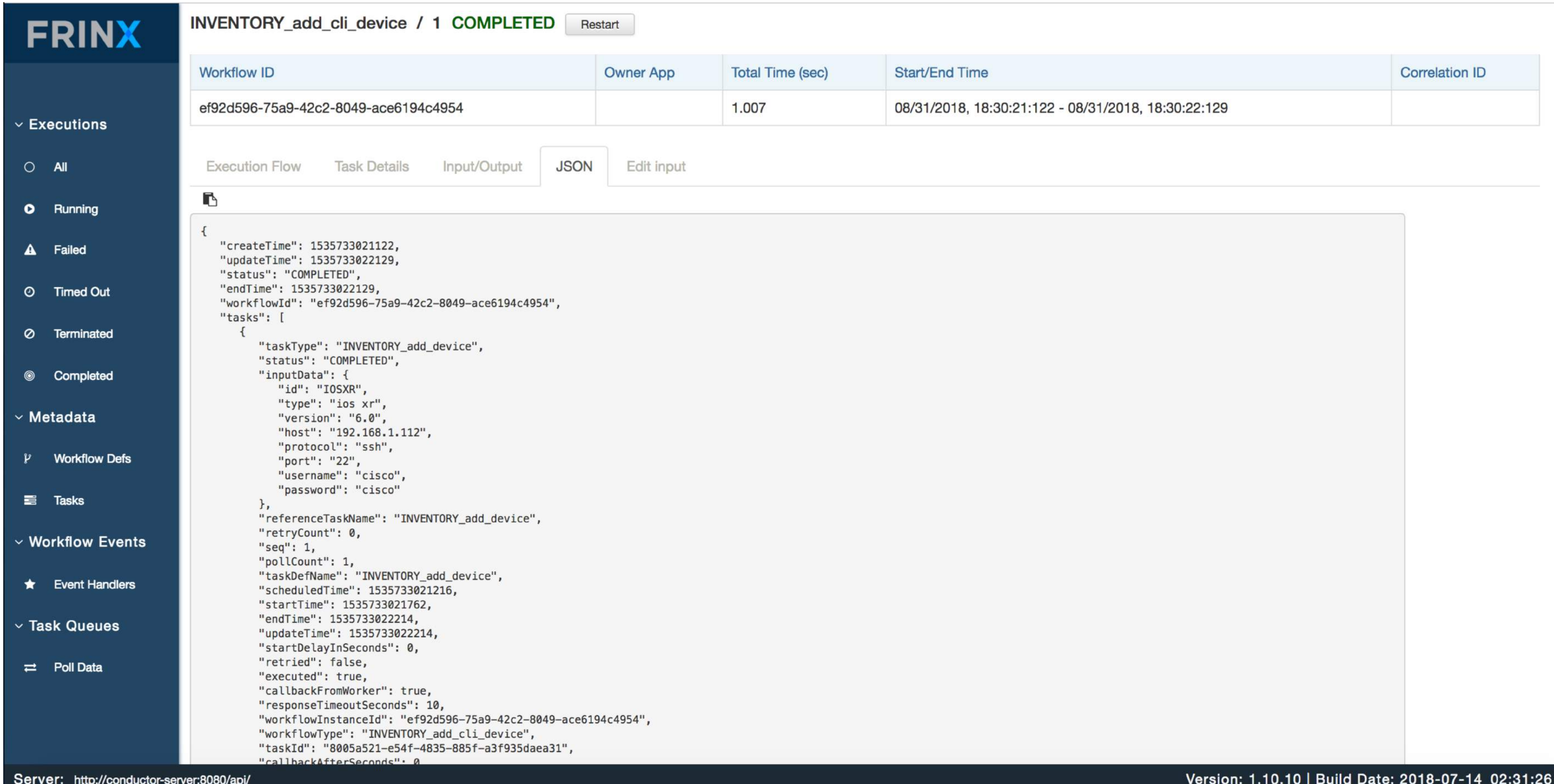




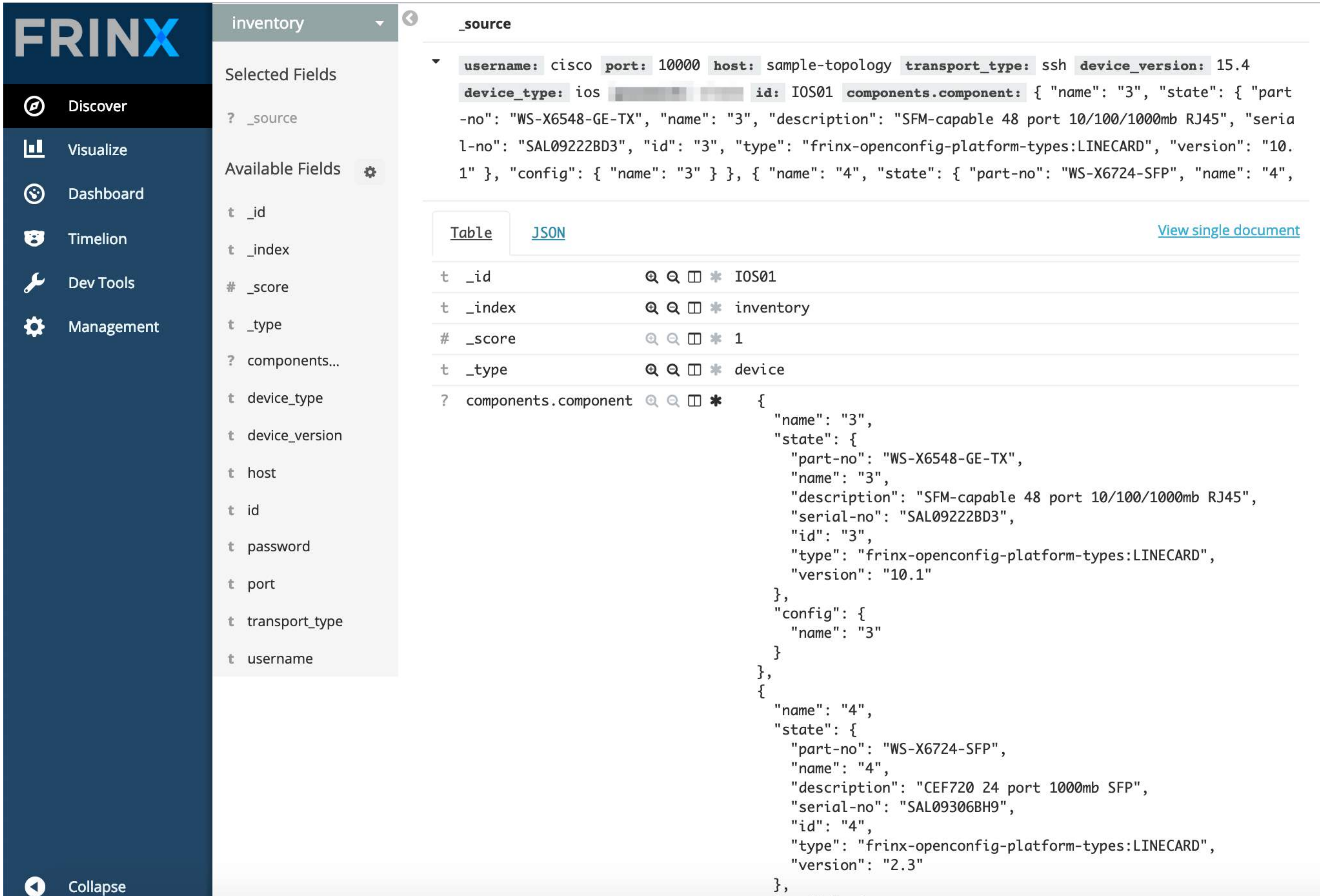
FRINX Workflow UI provides detailed task information (input and output data per task, failure reason, time stats)



FRINX Workflow UI provides detailed stats about task and workflow execution and status



FRINX Inventory UI



FRINX UniConfig Console

Good afternoon, **Gerhard**
gwieser@frinx.io

```
frinxit$ show odl version
Success. Status code: 200
{
  "output": {
    "versions": {
      "controller-version": "3.1.7.frinx"
    }
  }
}
frinxit$ show uniconfig calculate-diff
Success. Status code: 200
No diffs between config and operational datastore.
frinxit$
frinxit$
frinxit$ show cli operational
Success. Status code: 200
Node ID             Host IP             Host Status
ASR01_CA01_SJ04     sample-topology     connected
ASR01_NJ03_ISL01    sample-topology     connected
ASR01_NJ03_ISL02    sample-topology     connected
ASR01_NJ03_MPLW07   sample-topology     connected
ASR01_NY08_BKLN01   sample-topology     connected
ASR01_NY08_BKLN02   sample-topology     connected
IOS01               sample-topology     connected
IOS02               sample-topology     connected
frinxit$ show cli operational | grep NY
ASR01_NY08_BKLN01   sample-topology     connected
ASR01_NY08_BKLN02   sample-topology     connected
frinxit$
```

For more details about Frinx please contact frinx@frinx.io