Engineering Simplicity

Juniper Apstra Server and Security

Published

2024-09-03

Juniper Networks, Inc.
1133 Innovation Way
Sunnyvale, California 94089
USA
408-745-2000
www.juniper.net

YEAR 2000 NOTICE

END USER LICENSE AGREEMENT

Table of Contents

About This Guide

Juniper Apstra™ is a powerful automation and abstraction software solution tailored for data center network infrastructure. It is designed to assist both network architects and operators by automating the design, building, deployment, and operation of data center networks. This is achieved through Intent-Based Networking - an approach that aligns technology with the architect and operator's intentions for a network, driven by business expectations.

Network operators require the highest levels of security to ensure ongoing availability and reliable business processes. The Juniper Apstra architecture is designed to adhere to common industry requirements. Apstra also address customer-specific security needs as they arise.

1

CHAPTER

Overview

Overview

Juniper Apstra™ is a powerful automation and abstraction software solution tailored for data center network infrastructure. It is designed to assist both network architects and operators by automating the design, building, deployment, and operation of data center networks. This is achieved through Intent-Based Networking - an approach that aligns technology with the architect and operator's intentions for a network, driven by business expectations.

Network operators require the highest levels of security to ensure ongoing availability and reliable business processes. The Juniper Apstra architecture is designed to adhere to common industry requirements. Apstra also address customer-specific security needs as they arise.

2
CHAPTER

Components

Apstra Components

IN THIS SECTION

The following sections describe the elements that comprise a Juniper Apstra system.

Apstra Server

Lightweight, scalable VM appliance that serves as the centralized management/configuration system. Appliance simply requires IP management access to switches, typically via the OOB management network.

Apstra Device Agents

Software agents are created and installed to manage each switch or server under Apstra management. Agents are used to both receive and instantiate intent (configuration) from the centralized Apstra server and to collect analytics from each switch and send it to the Apstra server where that data is stored in the graphDB.

Network Operating System

A Network Operating System (NOS) is an OS that is primarily designed to support the functionality of an integrated multi-layer switch or router. Apstra as a multi vendor solution currently supports: Cisco NXOS, Arista EOS, JunOS, and SONiC.

Management Network

The Juniper Apstra server simply requires IP connectivity between itself and any devices under management. There are no distance, latency or delay requirements. An OOB Management Network is required. Apstra does not sit in either the data or control planes.

3

CHAPTER

Software Architecture Overview

Software Architecture Overview

Juniper Apstra has the following two major components in its system architecture:

- Apstra Server
- Apstra Device Agents

Every device managed by Apstra will require an Apstra agent installed on it. Apstra server and all the Apstra agents act as a distributed operating system.

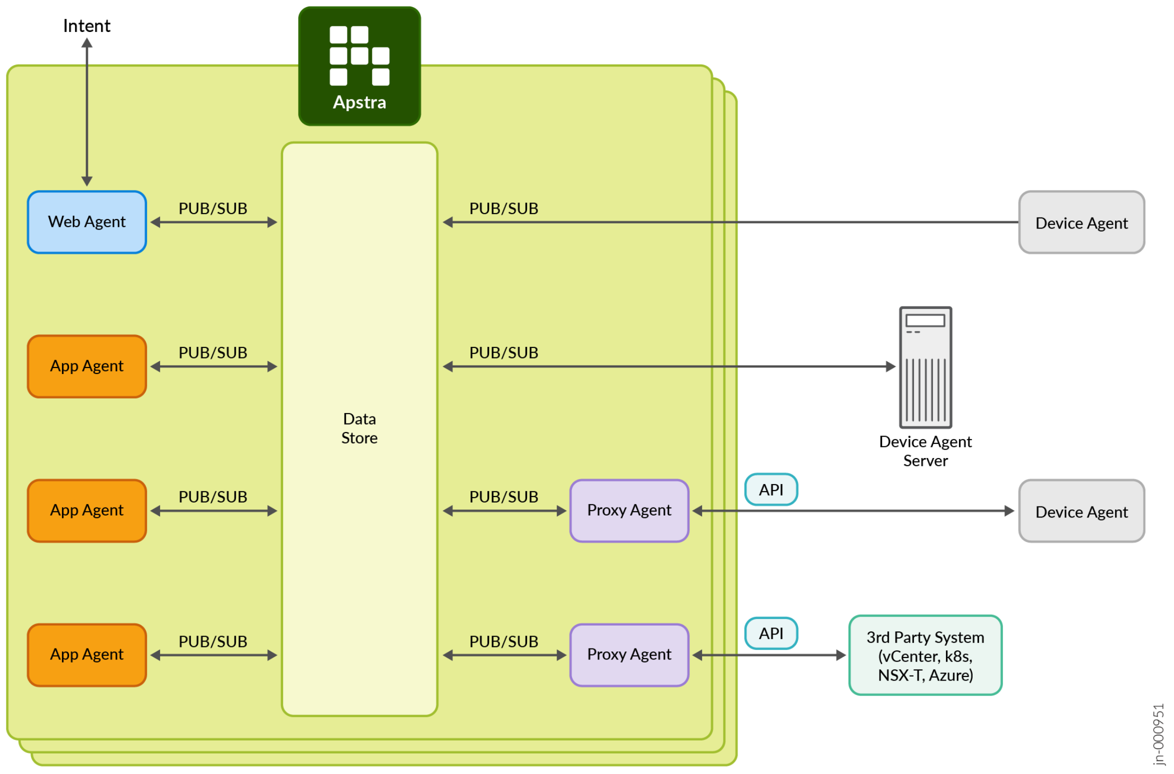
TCP connectivity is the only requirement between nodes.

Apstra translates high-level business requirements, referred to as "intent," and translates that into a fully operational data center network environment.

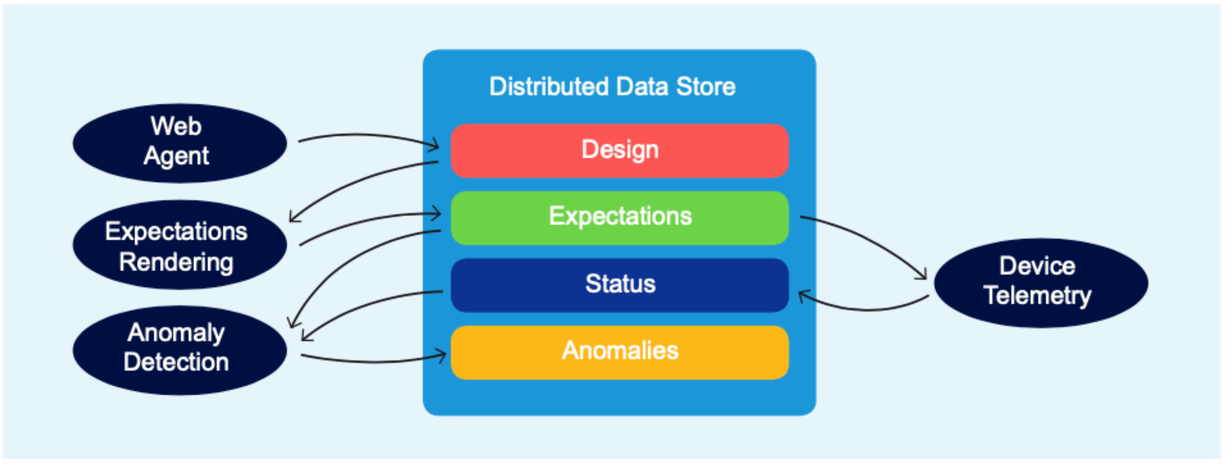
Juniper Apstra architecture is based on distributed state-management infrastructure, which can be described as a data-centric communication fabric with horizontally scalable and fault-tolerant in-memory datastore. All the functionalities of the specific reference design application are implemented via a set of stateless agents. Agents communicate with each other via a logical publish-subscribe-based communication channel and essentially implement the application's logic.

Every Apstra reference design application is simply a collection of stateless agents described above. Broadly speaking, there are three classes of agents in Apstra:

1. Interaction (web) agents are responsible for interacting with users, i.e., taking user input and feeding users with relevant context from the data store.
2. Application agents are responsible for performing application domain-specific data transformations, by subscribing to input entities and producing output entities.
3. Device agents reside on (or are proxies for) a managed physical or virtual system such as a switch, server, firewall, load balancer, or even controller and are used for writing configuration and gathering telemetry using native (device-specific) interfaces, often vendor's APIs.



This interaction can be illustrated with an example that describes a portion of the Apstra data center networking reference design application.



The web agent takes user input — in this case, a design for an L3 Clos Fabric that contains the number of spines, leafs, and links between them, and the resource pools to use for fabric IPs and ASN numbers. The web agent publishes this intent into the data store as a set of graph nodes and relationships and their respective properties.

The build agent subscribes to this intent and:

- Performs correctness and completeness validations
- Allocates resources from resource pools

Assuming the validations pass, the build agent publishes that intent along with resource allocations into the data store.

The config rendering agent subscribes to the output of the build agent. For each node, the config agent fetches the relevant data, including resources, and merges it with configuration templates.

The expectations agent also subscribes to the output of the build agent and generates expectations that need to be met in order to validate the outcome.

The device telemetry agent subscribes to the output of the expectations agent and starts collecting relevant telemetry. IBA probes process the raw telemetry and compare it against expectations and publish anomalies.

The Root Cause Identification (RCI) agent analyzes the anomalies and classifies them into symptoms, impacts, and identified root causes.

Agents communicate via attribute-based interfaces (hence the term data-centric) by publishing entities and subscribing to changes in entities. Data-centric also implies that data definition is part of the framework and is implemented by defining the entities, as opposed to message-based systems, for example.

The data-centric publish-subscribe system does not suffer from the problems of message-based systems. In a message-based system, sooner or later the number of messages exceeds the capacity of the system to store or consume them; dealing with this is hard as one must replay the history of messages to get to a consistent state. The data-centric system is resilient to surges in state changes as it is fundamentally dependent only on the last state. This state captures the important context and abstracts away all the possible (and irrelevant) event sequences that lead to it.

The difficult problems (e.g., elasticity, fault tolerance) are solved once, and on behalf of all agents. Typical architecture then consists of several stateless agents that can be restarted in case of failure and pick up where they left off by simply re-reading the state they subscribe to from the database.

4

CHAPTER

Apstra Server Overview

Apstra Server Overview

The Apstra Server is built using a microservices architecture and distributed as a packaged virtual appliance. The core application runs on Ubuntu Linux 18.04 LTS as a collection of seven major processes. Each process runs in a container to provide fast restarts and in-place upgrades of individual components. The Apstra Server images are automatically generated by the Apstra build system and tested against both physical and virtual switches in Apstra's system testing facility. For each Beta/GA build, a security scan is completed using Nessus Community Edition (more information about testing is included later in this document). Issues identified are either resolved/triaged or noted in the documentation and release notes. All release notes are accessible to customers at https://www.juniper.net/documentation/product/us/en/apstra/.

Customers can choose from either ESX, KVM or Microsoft Hyper-V hypervisors and should adhere to the vendor recommendations and best practices for securing management of the hosts.

5

CHAPTER

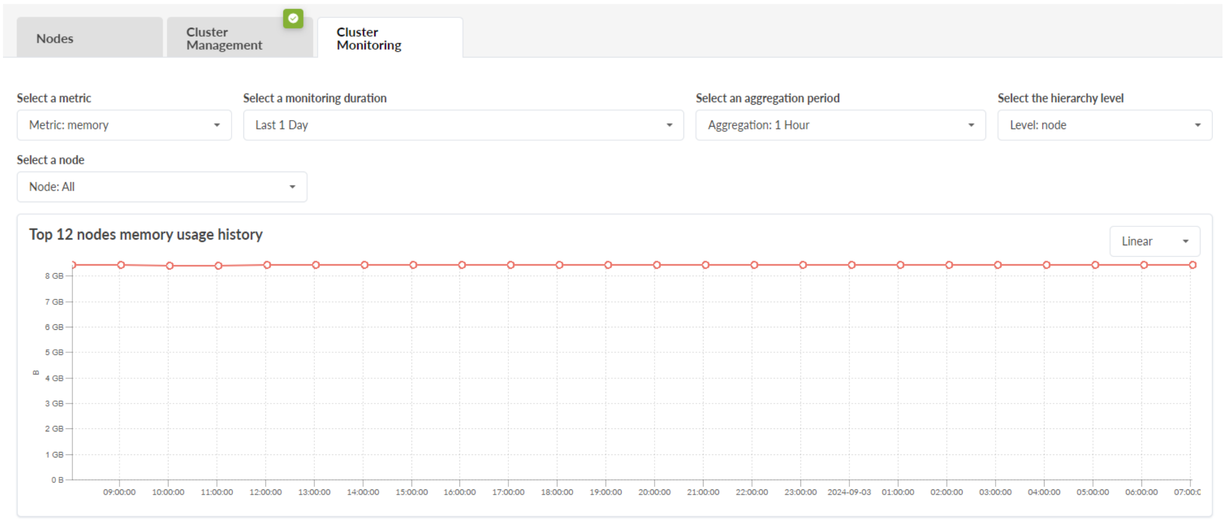
Recommended Apstra Server VM Resources

Recommended Apstra Server VM Resources

The required VM resources for an Apstra server may be greater than the recommendations below. Requirements are based on the size of the network (blueprint), the number of off-box agents, and the number of Intent-Based Analytics (IBA) probes. If one VM is insufficient for your needs, you can increase resources by clustering several VMs (Platform / Apstra Cluster). For more information about Apstra Server Clustering, see https://juniper.net/documentation/us/en/software/apstra4.2/Apstra-Server-Clustering-Guide/Apstra-Server-Clustering-Guide.html

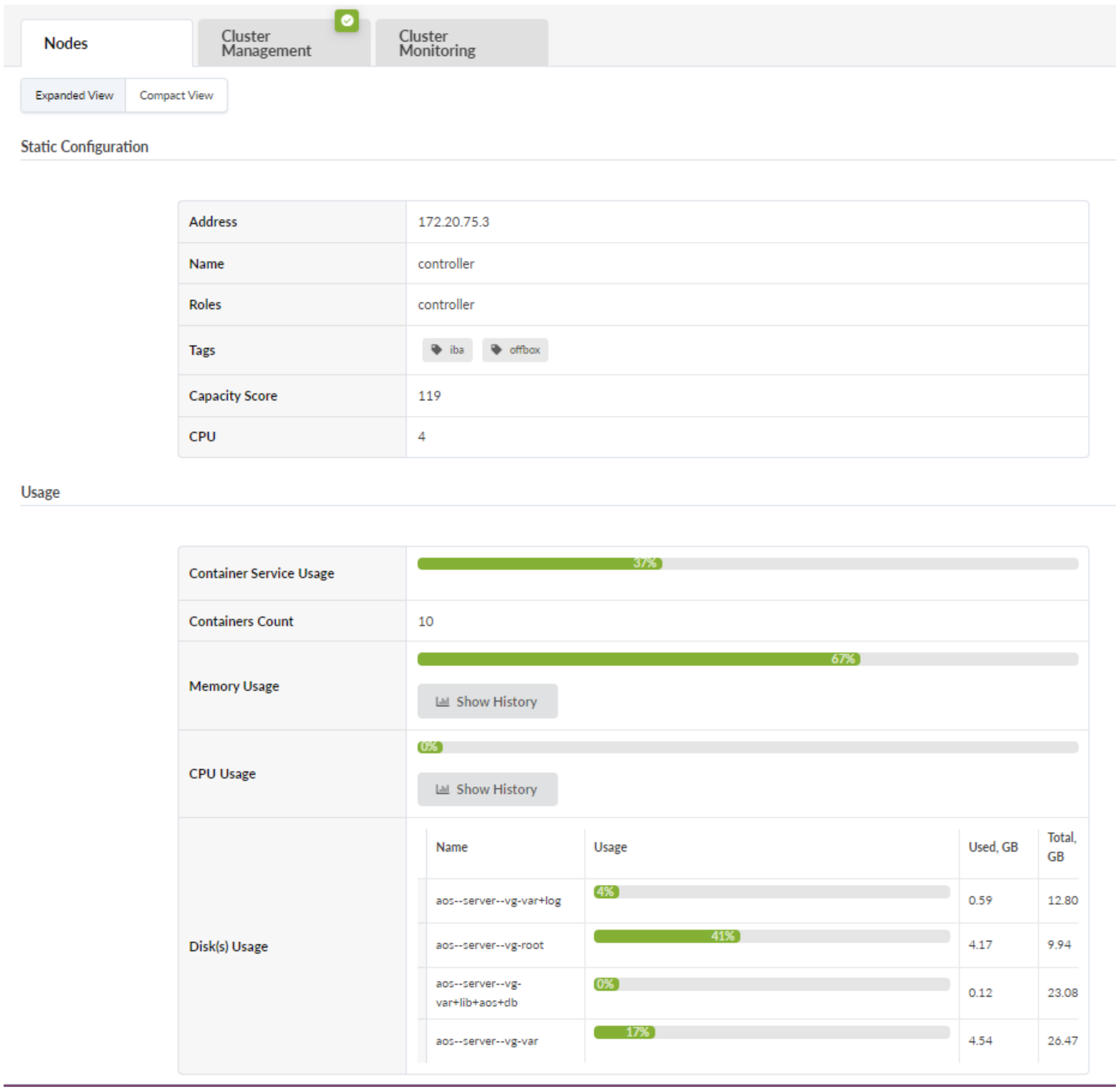
Resource	Recommendation
Memory	64 GB RAM + 300 MB per installed off-box agent*
CPU	8 vCPU
Disk	80 GB
Network	1 network adapter, initially configured with DHCP

NOTE: Apstra off-box agent memory usage is dependent on the number of IBA collectors enabled. We recommend that you use the Apstra UI to monitor memory/cpu usage in the Cluster Monitoring tab.



Select a metric: Metric: memory
Select a monitoring duration: Last 1 Day
Select an aggregation period: Aggregation: 1 Hour
Select the hierarchy level: Level: node
Select a node: Node: All

Top 12 nodes memory usage history



Nodes | Cluster Management | Cluster Monitoring

Expanded View | Compact View

Static Configuration

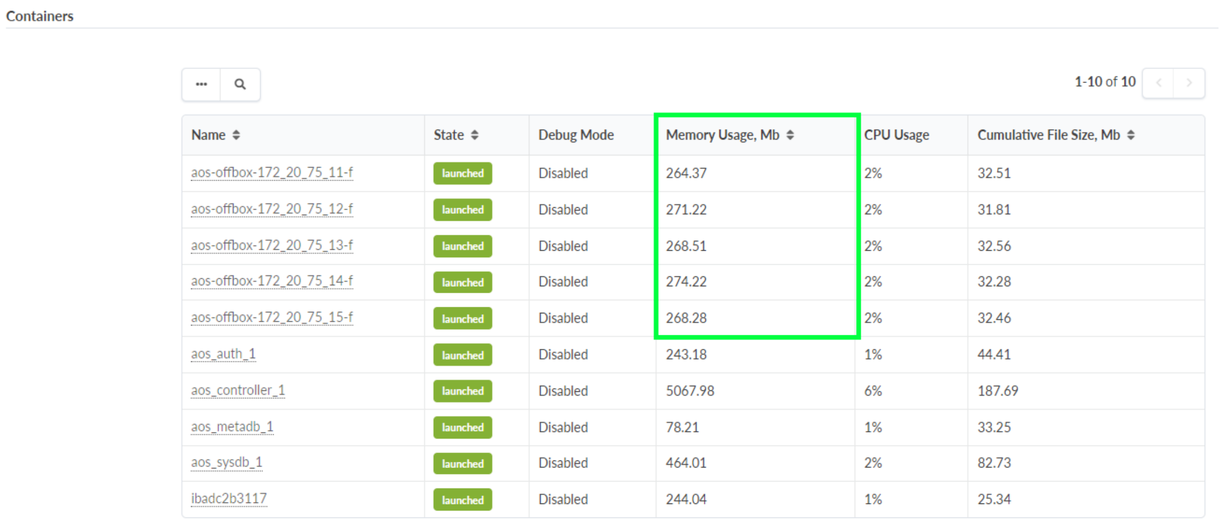
Address	172.20.75.3
Name	controller
Roles	controller
Tags	iba, offbox
Capacity Score	119
CPU	4

Usage

Container Service Usage	37%
Containers Count	10
Memory Usage	67% Show History
CPU Usage	0% Show History

Disk(s) Usage

Name	Usage	Used, GB	Total, GB
aos--server--vg-var+log	4%	0.59	12.80
aos--server--vg-root	41%	4.17	9.94
aos--server--vg-var+lib+aos+db	0%	0.12	23.08
aos--server--vg-var	17%	4.54	26.47


Containers

1-10 of 10

Name	State	Debug Mode	Memory Usage, Mb	CPU Usage	Cumulative File Size, Mb
aos-offbox-172_20_75_11-f	launched	Disabled	264.37	2%	32.51
aos-offbox-172_20_75_12-f	launched	Disabled	271.22	2%	31.81
aos-offbox-172_20_75_13-f	launched	Disabled	268.51	2%	32.56
aos-offbox-172_20_75_14-f	launched	Disabled	274.22	2%	32.28
aos-offbox-172_20_75_15-f	launched	Disabled	268.28	2%	32.46
aos_auth_1	launched	Disabled	243.18	1%	44.41
aos_controller_1	launched	Disabled	5067.98	6%	187.69
aos_metadb_1	launched	Disabled	78.21	1%	33.25
aos_sysdb_1	launched	Disabled	464.01	2%	82.73
ibadc2b3117	launched	Disabled	244.04	1%	25.34

NOTE: Although an Apstra server VM might run with fewer resources than specified above, the CPU and RAM allocations may be insufficient, depending on the size of the Apstra network. In this case, the system encounters errors or a critical “segmentation fault” (core dump). If this happens, delete the VM and redeploy it with additional resources.

RELATED DOCUMENTATION

Apstra Server Clustering Guide

6

CHAPTER

Containers

Containers

The Apstra Server application is made up of the following containers:

- **NGINX**: Provides user interface and REST API. Both methods of interacting with NGINX are request-based and are therefore stateless.
- **Metadb/Centraldb**: This container runs processes that manage agent lifecycle and metadata for the Apstra distributed pub-sub fabric.
- **Sysdb**: This container runs processes that manage the system running state. There are 3 processes: (1) MainSysDB hosts data for blueprints; (2) TelemetrySysDB hosts data for collected device telemetry; (3) CachacaSysDB hosts data for IBA. MainSysDB and CachacaSysDB data is persisted. When MainSysDB fails, its data is recovered from checkpoint and re-do logs from the filesystem. Device telemetry data and IBA data is periodically re-collected and re-computed.
- **Auth**: This container authenticates incoming requests, including all API requests. The server process may contact an external authentication source such as LDAP. The server process is the entry point of the container. If the server process dies, the container is restarted.
- **Controller Agent**: This container runs processes that run the business logic of Apstra - building blueprints, managing resource allocations, rendering device configurations, etc. These agents obtain input/output state from SysDB processes. They are written to be idempotent and restart-safe. In case of agent failure, the agent is restarted, gets the latest state from SysDB and continues processing from that point.
- **Offbox Agent**: Apstra spawns a container on the controller VM for each device managed in off-box mode. The container runs the device agents collecting telemetry from that device. They publish data to TelemetrySysDB.
- **Credential**: This container stores username/password (original, not hashed) needed by Apstra to authenticate with remote servers. e.g. login username/password for network switches we manage. It is currently used by offbox and onbox agent manager. An admin is not allowed to browse "Credential" sysdb using tools like "acons". This resource can only be accessed by internal Apstra processes.

```
admin@Apstra-server:~$ docker ps
CONTAINER ID        IMAGE                     COMMAND                 CREATED
STATUS              PORTS               NAMES
b2209f10d499        Apstra:3.3.0-730            "/usr/bin/Apstra_launch..."   3 weeks
ago         Up 3 weeks                          ibaf2c112cf
64d822b8903a        Apstra:3.3.0-730            "/usr/bin/Apstra_launch..."   3 weeks
ago         Up 3 weeks                          Apstra_sysdb_1
ae7ef4f91841        Apstra:3.3.0-730            "/usr/bin/Apstra_launch..."   3 weeks
```

```
ago         Up 3 weeks                            Apstra_auth_1
916b80178197        Apstra:3.3.0-730              "/usr/bin/Apstra_launch..."   3 weeks
ago         Up 3 weeks                            Apstra_metadb_1
44807f97d463        nginx:1.14.2-upload-echo   "nginx -g 'daemon of..."   3 weeks ago
Up 3 weeks                            Apstra_nginx_1
09a7205d3cd7        Apstra:3.3.0-730              "/usr/bin/Apstra_launch..."   3 weeks
ago         Up 3 weeks                            Apstra_controller_1
admin@Apstra-server:~$
```

7

CHAPTER

Scaling

Scaling

IN THIS SECTION

Growth Without Pain

Apstra supports network-transparent distributed state access and management, while parallel execution is supported by the separate processes. Real-time execution is supported by an event-driven asynchronous model of execution together with real-time scheduling of execution. Efficiency and predictability are supported by compilation through C++ as an intermediate language to achieve machine- level efficiency.

There are three dimensions to scaling:

- State
- Processing
- Network Traffic

Scaling the State

The Juniper Apstra data store scales horizontally, by adding more high-availability (HA) pairs of servers. In Apstra, intent and telemetry data stores are separated and can scale independently as needed.

Scaling Processing

Juniper Apstra can launch multiple copies of processing agents (per agent type) if and when required that will share the processing load. More agents can be added by adding more servers to host them, and an agent's lifecycle is managed by Apstra.

Apstra's state-based pub/sub architecture allows agents to react (provide application logic) to a well-defined subset of state. Coverage of the whole intent is done through separate agents delegated to dealing with different subsets of state. This means that when there is a change in the intent or operational state, the agent's reaction is to "incremental change" and is independent of the size of the whole state.

Apstra employs the traditional approach to deal with scale and associated complexity — decomposition. The "everyone knows everything" approach doesn't scale, so Apstra distributes the knowledge about the desired state and lets each agent determine how to reach that state. This eliminates the need for centralized decision making. Because of this, the Apstra Server is not considered a "controller". Apstra's support for live graph queries implies that clients such as UI can ask for exactly what they want and get exactly what they need and nothing more, allowing granular control of the amount of data to be fetched from the back end.

Scaling Network Traffic

The third dimension is scaling network traffic. Communication between the agents and data store is using an optimized binary channel, thus significantly reducing the amount of traffic compared to text-based protocols.

Fault tolerance is achieved by executing the Apstra application as multiple processes, possibly running on separate hardware devices connected by a network and separating the state from processing with support for replicated state and fast recovery of state.

Apstra has been tested with production deployments of 400+ switches. Apstra has completed internally testing of network fabrics comprising up to 1600 virtual devices. Physical fabric size limitations are based on vendor form factor and software limitations.

8

CHAPTER

Server Administration

Apstra Server Administration

IN THIS SECTION

There are three methods to administer the Apstra Server:

- Web UI
- SSH
- REST API

Apstra has an advanced RBAC system to define fine grained control of administrative duties. Information on the authentication and RBAC options is beyond the scope of this document and can be found in the Apstra standard documentation: https://www.juniper.net/documentation/product/us/en/apstra/.

Javascript Web UI and REST API

SSL is implemented using, but not limited to, AES 128/192/256 (CBC or GCM), RC4. For SSL key exchange, Apstra uses RSA with a key modulus up to and including 2048-bits and Diffie-Hellman with a key modulus of up to and including 2048-bits for key exchange.

SSH

SSH is implemented using 3DES, Blowfish, Twofish, CAST-128, IDEA, and ARCFOUR. For SSH key exchange, Apstra uses RSA with a key modulus up to and including 2048-bits and Diffie-Hellman with a key modulus of up to and including 2048-bits for key exchange.

APIs

Apstra abstracts individual networking hardware and software and presents users with easy-to-parse industry standard REST APIs which model the entire system state and support the JSON data format which has a number of ways of keeping specification or configuration in text files:

```
name: “Create Virtual Network”
Apstra_blueprint_virtnet:
  session: “{{ Apstra_session }}”
  blueprint: “my-blueprint-l2”
  content: “{{ lookup(‘file’,
  ‘resources/virtual-network-05.json’) }}”
  state: present
```

In summary, Juniper Apstra is a powerful tool for traditional networking engineers looking to improve their automation and programming skills. Instead of focusing on low-level network device actions, you can quickly interact programmatically with Apstra using declarative API as easy as you would any cloud-based solution with tools like Ansible, Postman, or automation tool of choice.

9

CHAPTER

Device Agents

Apstra Device Agents

The Apstra Device Agents function in one of two ways: Onbox or Offbox.

Option 1: Apstra Agent installed on the switch (Onbox)

In this case the Apstra Agent has been installed onto the device either via the ZTP boot process or using the Apstra Device Installer. Once the agent is installed it will always run, including after reboots. Communication between the agent and the Apstra Server are done over a highly optimized binary protocol.

The Apstra Agent package installs the following processes within the Network Operating System (NOS) namespace to create an isolated runtime environment:

- **Counter Agent:** Responsible for retrieving counters from a device and sending them upstream to the Apstra Server. The majority of traffic is normally generated by this agent.
- **Deployment Agent**: Responsible for accepting configuration pushed down from the Apstra Server and applying it to the device. This agent is idle most of the time.
- **Telemetry Agent:** Responsible for retrieving LLDP, routing, interface information and other telemetry and sending it upstream to the Apstra Server. This agent is idle most of the time, except when important events occur.
- **Local Process Spawner:** Responsible for instantiation of the agent.
- **Local SysDB**: Each device maintains a localized version of the SysDB process to store intent for local purposes.

The port used to connect to devices can be adjusted in the Apstra server. The default ports for this protocol are:

Agent <==> MetaDB (TCP dst port 29731)

Agent <==> SysDB (TCP dst port 29732)

Agent <==> CentralDB (TCP dst port 29730) (future)

Agent ⇐=> TelemetrySysDB (TCP dst port 29733)

The Apstra agents are installed inside of a protected guestshell or userland in each vendor device. The agent processes are isolated from the underlying switch hardware and software, Apstra does not directly talk to the forwarding/data plane or control plane.

Option 2: Apstra Proxy Agent (offbox) connects to the device via the vendor's standard API or CLI/SSH

The Proxy Agent makes connections on the defined API port (typically 80/443/9443) or standard SSH (typically 22). Connections are initiated by the proxy agent and this happens on a set time interval or when updates occur in Apstra. This agent runs as a container directly on the Apstra server.

Apstra implements SSH to secure management data between the product management interface. This product makes use of the SSH protocol using 3DES, Blowfish, Twofish, CAST-128, IDEA, ARCFOUR.

SSL/SSH Key Exchange

For the SSL and SSH implementations this product uses RSA with a key modulus up to and including 2048-bits and Diffie-Hellman with a key modulus of up to and including 2048-bits for key exchange.

10

CHAPTER

Agent Network Security

Apstra Agent Network Security

Apstra communicates with the management interfaces of devices via the IP address of the dedicated management port. Customers typically connect the management ports to an out-of-band (OOB) network that is restricted to network engineers, NOC management systems, and an emergency “break glass” IP subnet via ACLs or firewalls.

NOTE: Currently, it's not possible to manage devices via the fabric ports.

11

CHAPTER

Server Hardening

Apstra Server Hardening

IN THIS SECTION

Vulnerability Scanning

A Tenable / Nessus vulnerability scan occurs weekly against all released, supported versions of Juniper Apstra in addition to the latest build of any version in development. This software is set up to notify Apstra Engineering and Support for any newfound security vulnerabilities over a particular threshold of CVSS > 8.0. As part of these security findings, Juniper Apstra formulates a targeted plan to update security packages in the field.

Juniper Apstra uses an automated CI/CD process for building releases that include bringing in the latest versions of Ubuntu and updated packages. The SecOps part of the build system then runs a Tenable I.O Nessus scan for vulnerabilities. We review that report and fix any critical issues and review less critical problems to cherry-pick those vulnerabilities that we deem that require attention.

Our only path to resolving security issues is a software upgrade where the new Ubuntu release and related packages, bundled with the Apstra Software, have been tested and placed into longevity tests for any regression, performance, or functionality issue. The version of Ubuntu and packages are pristine when we build and ship them. Still, vulnerabilities discovered over time will result in additional problems that are resolved by the upgrade. For isolated critical issues, we can create and apply a hotfix to a production deployment or recommend a mitigation procedure.

Process for Hardening (In-Build Process)

Before the customer downloads the Apstra VM, Apstra hardens it against the current CIS Ubuntu version as a "Level 1" system. This process includes basic security configuration in a few security domains, summarized here. The VM has the latest versions of all Ubuntu packages as of the date the Apstra release is available minus six weeks (validation and burn-in time before GA).

The Apstra VM Image is built off a multi build phase process:

1. Apstra creates a base Ubuntu system image from the Ubuntu ISO image from Ubuntu repositories online.
2. Validation of the authenticity of source ISO with SHA hashes
3. Apply minimum configuration and minimum installation packages for Apstra to install and automatically assign security policies. The configuration used is similar to the CIS Ubuntu hardening standards, https://www.cisecurity.org/benchmark/ubuntu_linux/, which aligns with many security compliance frameworks in the enterprise space.
4. After hardening scripts run, build artifacts, and cache folders are deleted to minimize disk usage, log files trimmed, and base system VM is 'closed'.
5. As part of the official build release artifacts, we reuse the base system created above installing only the Apstra-specific packages.
6. Build artifacts for the same, raw VM image is converted into OVA for VMware, QCOW2 for Linux KVM, and VHDX for Microsoft Hyper-V and marked as ready for distribution (https://support.juniper.net/support/downloads/?p=apstra)

   .

Hardening/Validation Steps

- Source validity
  - Ubuntu source ISO is validated with SHA hashes before the build process begins
- HTTPS Encryption
  - The Apstra Server provides the Apstra UI and API using HTTPS on Nginx. Apstra uses Nginx documented procedures to replace TLS certificates in order to accomplish client to server authentication and end-to-end encryption. Apstra encourages users to replace the included insecure, "self-signed" TLS certificates with signed certificates.
- File system security
  - Disable unnecessary file system drivers
  - Ensure file system mounts are secure
  - Apstra services don't share disk space with rest of OS (e.g./var/log/Apstra)
  - File system permissions are strong

- Services security
  - Disable unnecessary services
  - Configure AppArmor
  - Harden SSH configuration
- Network security
  - Ensure iptables is configured with an appropriate 'default deny' policy
  - Ensure the system has uRPF, disable IP routing, etc
  - Warning banners that state that any action should only be made with support from Apstra.
  - Apply login throttles to ssh login attempts (brute force prevention)
- Auditing
  - Ensure system configuration changes are audited (run auditd)
  - Audit login/logout failures and successes
  - Audit privilege escalations
- Secure server access
  - There are three methods to administer the Apstra Server:
    1. Javascript Web UI (443): HTTPS, Basic Authentication and RBAC*
    2. REST API (443): HTTPS, Basic Authentication and RBAC*
    3. SSH (22): No root login permitted by default

NOTE: IMPORTANT: Change the default admin password. Apstra ships with a default admin password. It is critical that this password is changed after the server has completed the first boot process. Apstra recommends using SSH to access the Apstra server to change the password. For more information, see Reset Apstra GUI Admin Password.