Scenarios - Scenario - secure analytics platform

[PDF generated October 31 2024. For all recent updates please see the Nutanix Bible releases notes located at https://nutanixbible.com/release_notes.html. Disclaimer: Downloaded PDFs may not always contain the latest information.]

In this scenario we will design a secure analytics platform ingesting data from various external and internal sources and ETL'ing into a data lake where analytics are performed.

Given the scope of this being very large and proprietary, only the security aspects of the platform will be discussed in the first iteration.

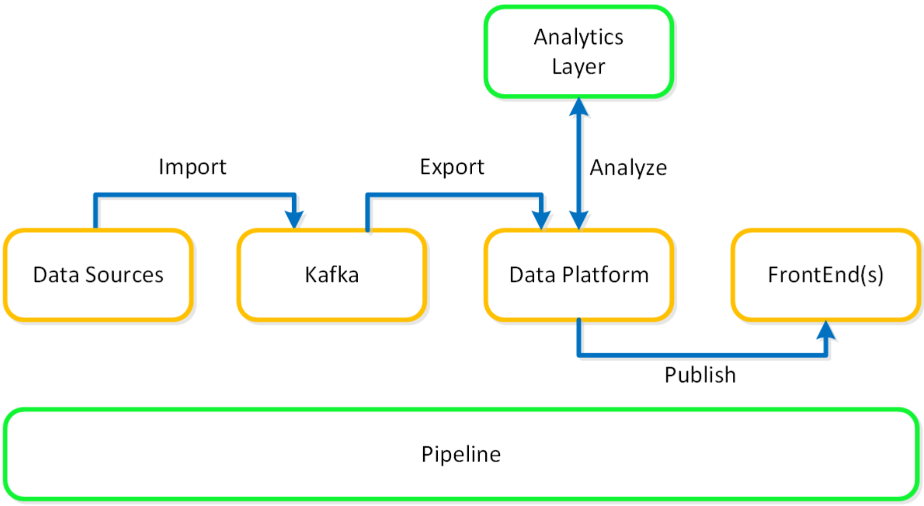
At a Glance

- Key Requirements
    - Environment must be highly secured through all layers (network, application, etc.)
    - Access must be scoped to specific enclaves / users
    - Must consume data from both internal and external sources
    - Configuration must be automated
    - User management / RBAC must be 100% automated
    - Data must be encrypted

- Solution Components
    - Front-End: Tableau
    - Service Catalog: Service Now
    - Approvals: Slack / Email -> ServiceNow
    - Configuration Automation: Puppet
    - Database: Apache MySQL / extensible
    - ESB / ETL: Custom + Apache Kafka
    - Analytics: Custom

- Platform
    - Hypervisor: Nutanix AHV
    - Encryption: Nutanix software encryption
    - Microsegmentation: Nutanix Flow
    - Compute + Storage + Network (virtual): Nutanix

The following shows a high-level view of the solution layers and data flows:



Scenario - Secure Analytics Platform

Security Architecture

As mentioned in the 'Security and Encryption' section, security occurs at multple levels ranging from data to systems to people. In the following we will cover how we hardened each of these components at a high-level.

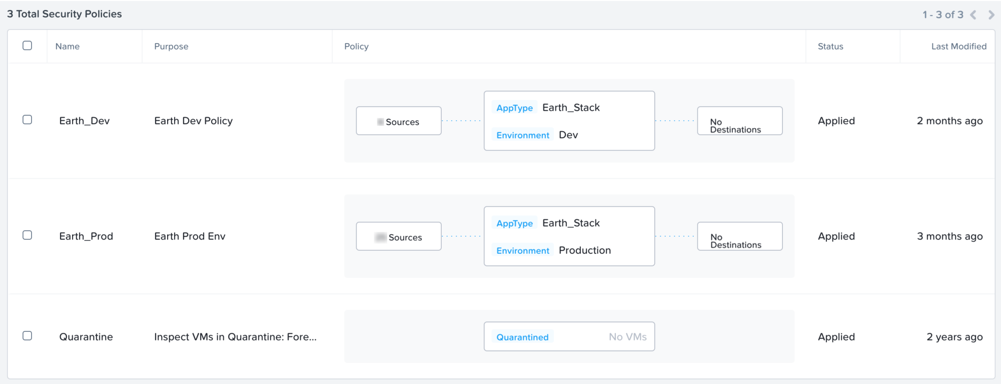
Networking & Communication

When it comes to networking and communication we need to ensure only known / secure enclaves were able to get access to the systems and data flows outbound are restricted.

We achieved this using a few items in alignment:

- All configurations are 100% automated using Puppet
- All policies are whitelist only
- Only truted enclaves are allowed inbound on specific ports
- All developer access flowed through a single jump box
- MySQL users / grants were scoped to specific user / IP addresses combinations
- Firewalld rules secured the Linux firewall
- Nutanix Flow secured the virtual / physical network layer

The following shows the Flow policies for the dev/staging/production environments:



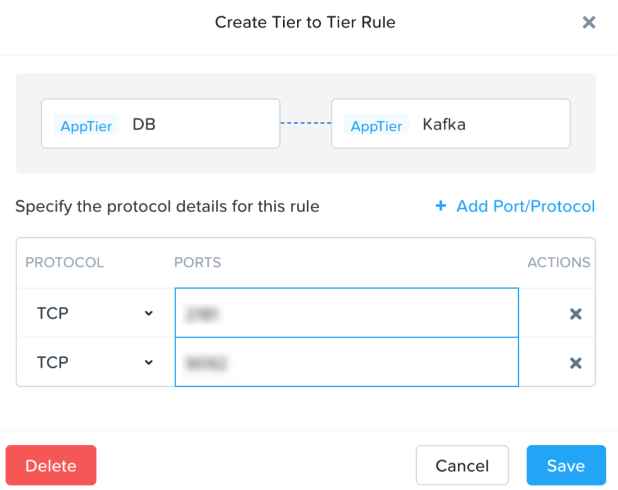
3 Total Security Policies

1 - 3 of 3

□	Name	Purpose	Policy	Status	Last Modified
□	Earth_Dev	Earth Dev Policy	[redacted] Sources AppType Earth_Stack Environment Dev No Destinations	Applied	2 months ago
□	Earth_Prod	Earth Prod Env	[redacted] Sources AppType Earth_Stack Environment Production No Destinations	Applied	3 months ago
□	Quarantine	Inspect VMs in Quarantine: Fore...	Quarantined No VMs	Applied	2 years ago

Scenario - Secure Analytics Platform - Flow Policies

Only specific ports / protocols were allowed between application tiers and inbound:



Create Tier to Tier Rule

AppTier DB

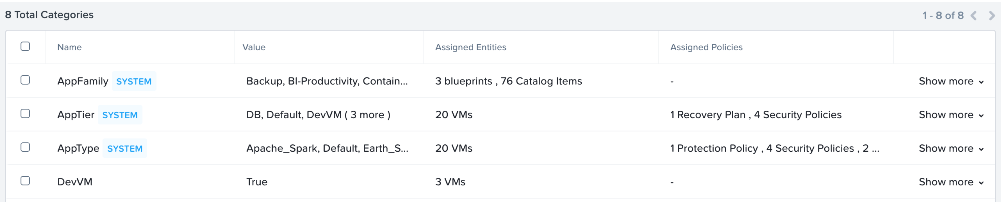
AppTier Kafka

Specify the protocol details for this rule

PROTOCOL	PORTS	ACTIONS
TCP	[redacted]	×
TCP	[redacted]	×

Scenario - Secure Analytics Platform - Flow Policy Detail

Categories were leverated to specify app tiers and environments. Only certain ports were allowed between:

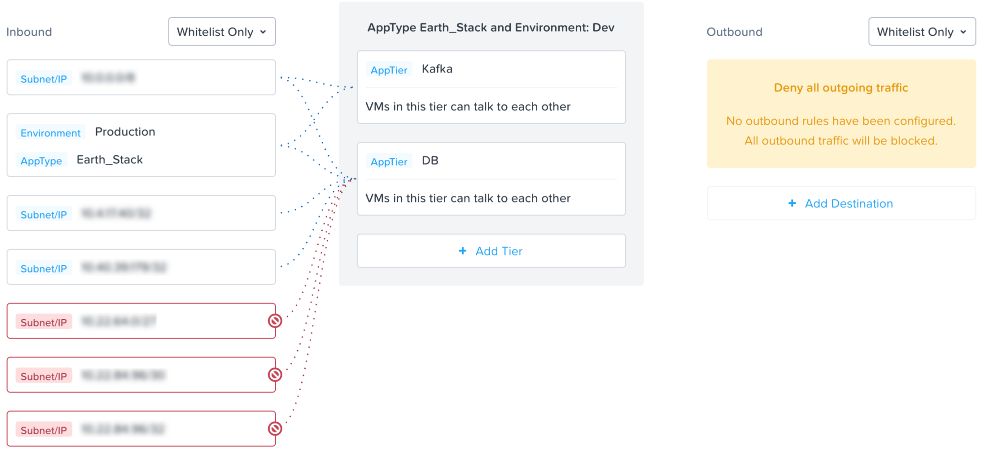


8 Total Categories 1 - 8 of 8

□	Name	Value	Assigned Entities	Assigned Policies	
□	AppFamily SYSTEM	Backup, BI-Productivity, Contain...	3 blueprints , 76 Catalog Items	-	Show more ⌄
□	AppTier SYSTEM	DB, Default, DevVM (3 more)	20 VMs	1 Recovery Plan , 4 Security Policies	Show more ⌄
□	AppType SYSTEM	Apache_Spark, Default, Earth_S...	20 VMs	1 Protection Policy , 4 Security Policies , 2 ...	Show more ⌄
□	DevVM	True	3 VMs	-	Show more ⌄

Scenario - Secure Analytics Platform - Policy Categories

Here's a sample look at a Flow policy for dev which shows the allowed inbound sources. It also highlights the blocked connections which coincidentally were from an internal pentesting tool:



Inbound Whitelist Only ⌄

Subnet/IP [redacted]

Environment Production
AppType Earth_Stack

Subnet/IP [redacted]

Subnet/IP [redacted]

Subnet/IP [redacted]

Subnet/IP [redacted]

Subnet/IP [redacted]

AppType Earth_Stack and Environment: Dev

AppTier Kafka
VMs in this tier can talk to each other

AppTier DB
VMs in this tier can talk to each other

+ Add Tier

Outbound Whitelist Only ⌄

Deny all outgoing traffic

No outbound rules have been configured.
All outbound traffic will be blocked.

+ Add Destination

Scenario - Secure Analytics Platform - Flow Policy Detail

Systems and Configuration

When it comes to the stack there were a few core layers:

- Application / Services
- VMs / Containers
- Infrastrucutre (Nutanix)

The full stack was 100% automated using Puppet, Nutanix SCMA and environment templates. This allowed us to ensure security / configuration baselines and adherance to them. This also allowed us to simply update packages if any security vulnerabilities were found.

Within the Nutanix platform the native SCMA was leveraged (enabled by defualt) which ensures a STIG'd / secure configuration for the CVMs and hosts. Cluster lockdown mode was enabled (recommended) to force key based access.

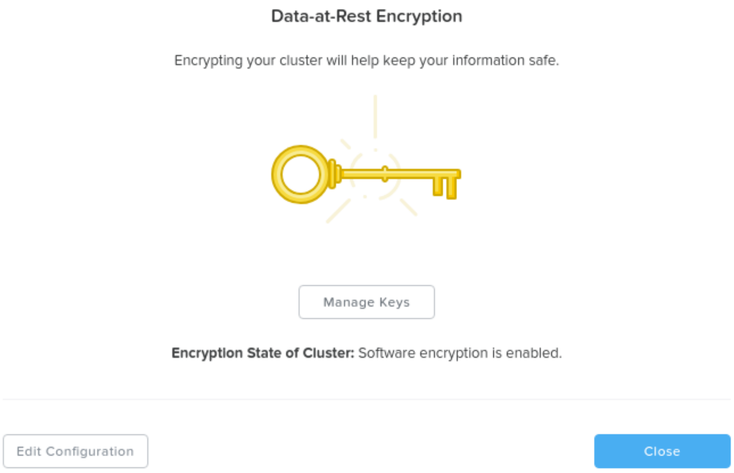
Secrets

With any platform that is integrating with multiple systems, secret management is a very important item. Initially we started with using encrypted yaml (eyaml) within Puppet but eventually moved this to a more secure / manageable hiera backend. There are multiple options here like HashiCorp Vault, etc.

Data Encryption

Data encryption is important to ensure an attacker can't make any sense of the data if they were to steal or become in posession of it.

Native Nutanix software-based encryption was leveraged to provide data encryption. In scenarios where a key manager isn't available, the local key manager (LKM) can be leveraged. If using an external key manager (EKM) it is recommended to rotate keys, this occurs yearly with the LKM by default.



Data-at-Rest Encryption

Encrypting your cluster will help keep your information safe.

Encryption State of Cluster: Software encryption is enabled.

Scenario - Secure Analytics Platform - Data Encryption

Data Scoping and RBAC

One of the most important things once the "stack" has been hardened, is ensuring only specified individuals get access to the data they should have access to and all access / requests / granting is fully auditible. From my perspective the only way to accurately do this is through a fully automated system where any granting of access, approvals, etc. has to flow through the automation and nothing can be done manually. This is exactly what we did with Project Earth, even as admins, we can't override access for users.

If we break this down there are a few core stages:

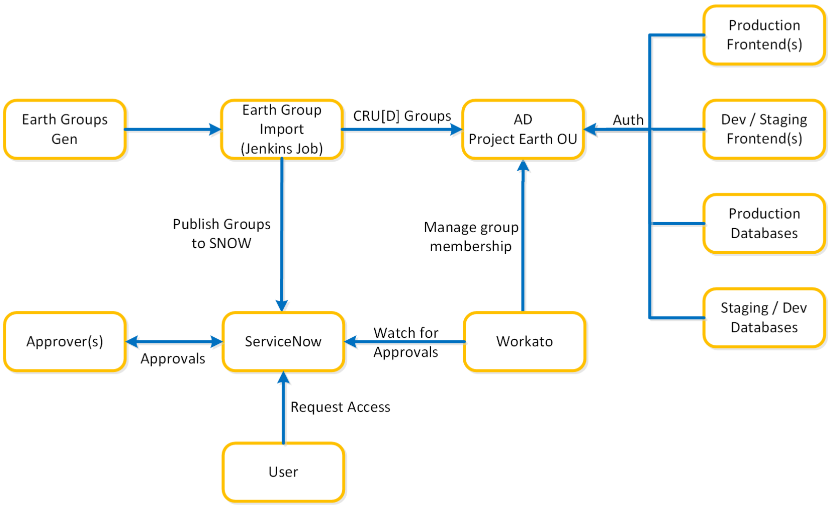
- Requesting / inheriting access
- Approving / denying access
- Validation & Q/A
- Revocation
- Auditing

For all requests and ticketing we leverage ServiceNow (SNOW). Within SNOW we created a custom catalog in which users could request access to specific types of data. The available types of "roles" / data available are automatically generated and published to SNOW whenever new data sources / roles are created.

Once requested, it would go to their manager for approval and then two other approvers who must approve before any access is granted. This "dual key" approval ensured proper checks and balances. Another key point here is that the time which people could request access for was limited and could be revoked at any time. Upon expiration / revocation, membership was automatically removed.

Once the request was approved the role assignment / group membership was fully automated.

The following figure shows a high-level view of the flow:



Scenario - Secure Analytics Platform - RBAC / Role Assignment

For validation we have checks to ensure members of a group match the "approved" state in SNOW. All authentication / access requests are logged and stored in a central logging system. Using this system we could look for anomalous access or things out of the ordinary.

Change Control

A key for auditibility is proper change control throughout all aspects of the system. For requests / approvals those were all stored in SNOW. All other items whether it be Puppet, custom logic and code, etc. were all kept in a hardened source control system with only specific developers / keys have access. Any modifications/ new code first went through a code review process / security validation. Once reviewed / approved the changes were put into "purgatory" until validation in dev / staging environments were coplete. Any modifications to production systems are done using automation to ensure human error potential is minimized.