COPS Platform

Table of Contents

Overview

This page describes an overall design for the Connected Operations (COPS) Platform.

The COPS Platform provides a suite of tools for hosting RESTful services through a single interface while isolating them at different security levels. It provides a mechanism for securing user access, storing data using SELinux labels, and running services in isolated containers according to SELinux labels.

One or more COPS Platforms may be present in a system, and each one can handle data across different security levels and categories. Furthermore, each component could be running within the same Operating System (server, VM, instance, etc), or they could be spread across different computers.

More details appear later in this document, but figure 1 provides an overview of the components within the COPS Platform. Lower-level details (and relevant research into related products) are on subpages and the COPS Platform Components.

Consumers request services through the Platform Controller. Those services may allow things like:

- Submitting data for storage with specific SELinux sensitivities and categories
- Retrieving data based on the user's authorization and the SELinux labels on the requested data
- Viewing reports that a service has compiled based on an aggregation of data from multiple sources

Note also that the Platform Controller + IdAM may be sufficient in some configurations (without the Container Runtime and/or Data Storage Components). For example, a trusted application may store its own data internally, and the Platform Controller could still handle consumer connections and user authentication.

Figure 1: COPS Platform Components and how they communicate

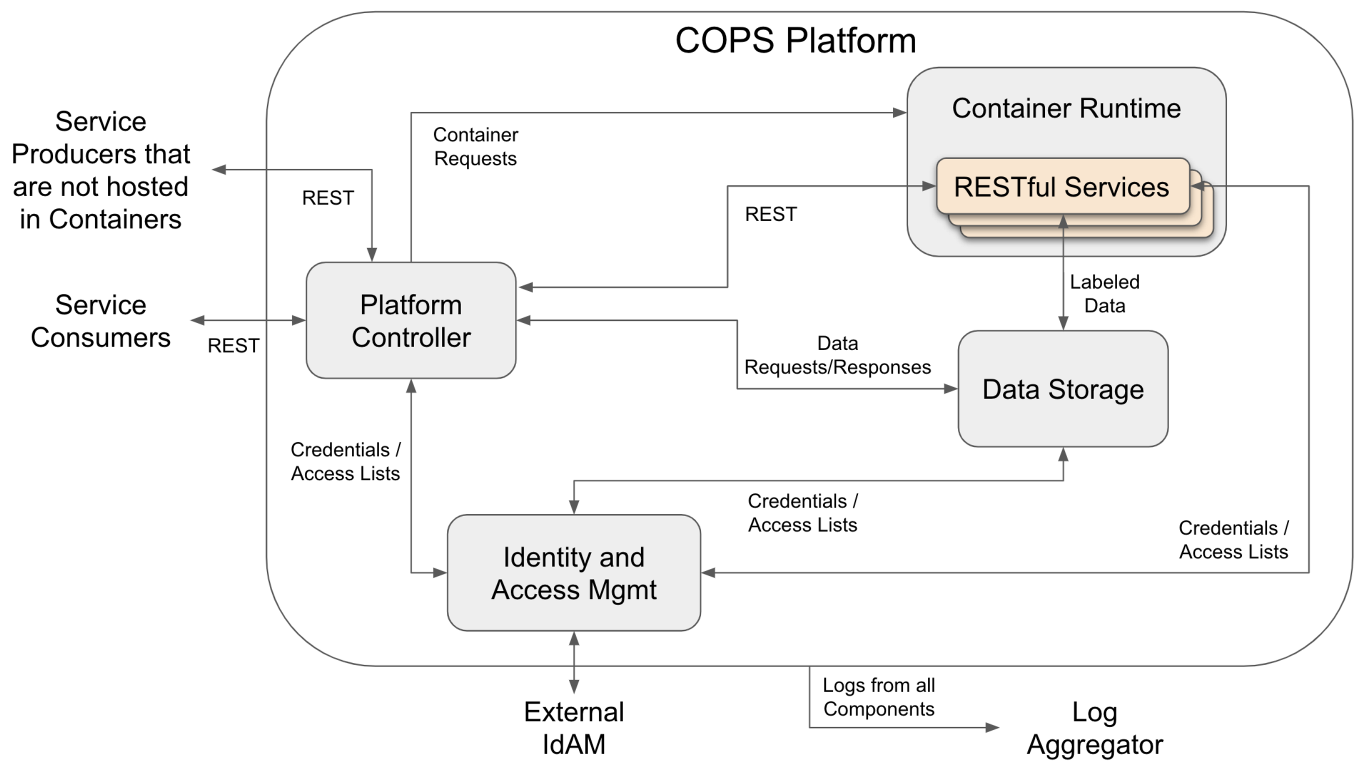


Figure 2 shows the basic workflow that uses these components to secure access to services and data. The "External Consumer" initiates the flow, and their access is validated using a combination of their User Credentials and the identity of the system from which they connect. The service they interact with is instantiated with their specific accesses within an isolated container.

For services that are not hosted internally within the container runtime, the Platform Controller can still serve as a central access point by forwarding requests to external services. The same flow still applies, but all steps involving the Container Runtime are skipped.

Figure 2: Flow through the COPS Platform Subsystem

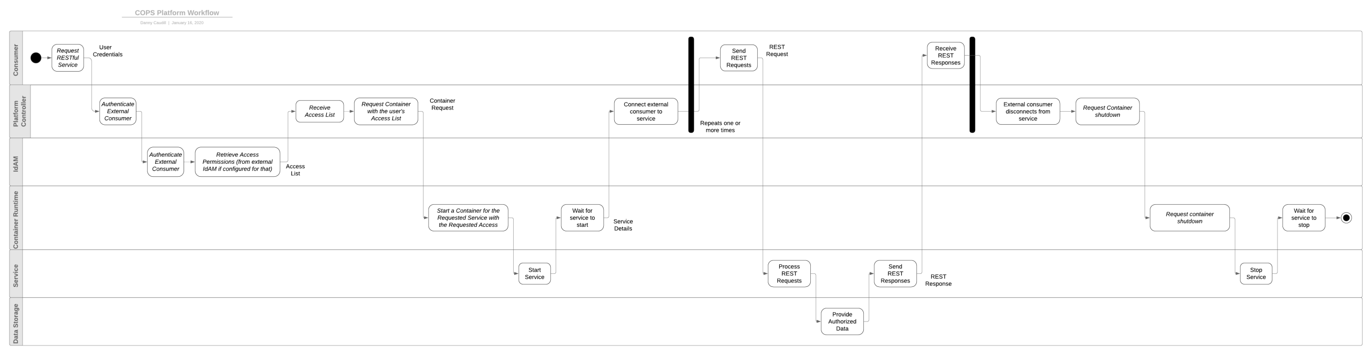
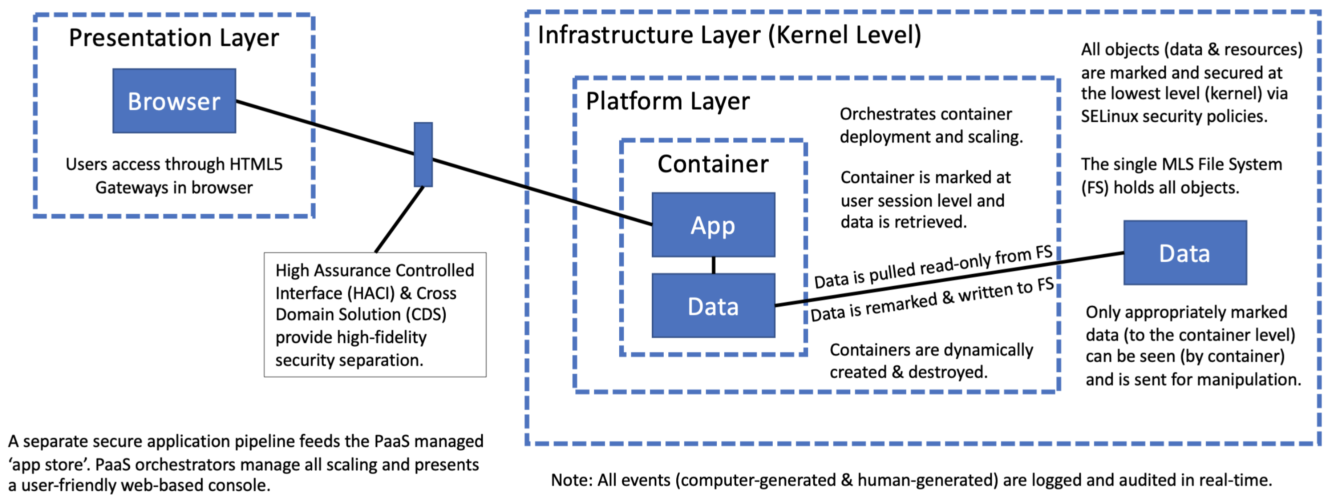

COPS Platform Workflow
Danny Caudill | January 16, 2020
Consumer
Platform Controller
IdAM
Container Runtime
Service
Data Storage
Request RESTful Service
User Credentials
Authenticate External Consumer
Authenticate External Consumer
Retrieve Access Permissions (from external IdAM if configured for that)
Access List
Receive Access List
Request Container with the user's Access List
Container Request
Start a Container for the Requested Service with the Requested Access
Start Service
Wait for service to start
Service Details
Connect external consumer to service
Repeats one or more times
Send REST Requests
REST Request
Process REST Requests
Provide Authorized Data
Send REST Responses
REST Response
Receive REST Responses
External consumer disconnects from service
Request Container shutdown
Request container shutdown
Stop Service
Wait for service to stop

Figure 3 shows the original overview for a Multi-Level Security architecture, upon which this design is based. The COPS Platform represents the Platform Layer in this diagram, with the addition of data storage.

Figure 3: Original MLS architecture overview

MLS/MAC Cloud Security Model Overview


Presentation Layer
Browser
Users access through HTML5 Gateways in browser
High Assurance Controlled Interface (HACI) & Cross Domain Solution (CDS) provide high-fidelity security separation.
A separate secure application pipeline feeds the PaaS managed 'app store'. PaaS orchestrators manage all scaling and presents a user-friendly web-based console.
Infrastructure Layer (Kernel Level)
Platform Layer
Container
App
Data
Orchestrates container deployment and scaling.
Container is marked at user session level and data is retrieved.
Data is pulled read-only from FS
Data is remarked & written to FS
Containers are dynamically created & destroyed.
All objects (data & resources) are marked and secured at the lowest level (kernel) via SELinux security policies.
The single MLS File System (FS) holds all objects.
Data
Only appropriately marked data (to the container level) can be seen (by container) and is sent for manipulation.
Note: All events (computer-generated & human-generated) are logged and audited in real-time.

Capabilities and Requirements

The COPS Platform Shall:

- Host RESTful services internally within containers
- Provide a central access point for services hosted internally and external services
- Provide Multi Level Data Storage in the form of SQL databases
- Provide isolation (encryption and access control) between security levels of:
  - Data at rest
  - Compute resources
  - Data transiting through the Platform Controller
- Authenticate users via an external IdAM system
- Validate user authorization for access before granting access to individual APIs
- Provide a secure API Gateway through which RESTful services can be accessed by authenticated users
- Provide a secure Data API through which the Secure Data Storage may be accessed
- Work seamlessly with the selected Cross Domain Solution (CDS)
- Pass SELinux labels alongside all data traveling through the system
- Use SELinux labels for isolation whenever possible
- Synchronize all Operating System clocks to a common time server via the Network Time Protocol (NTP) so that log timestamps are meaningful
  - Configure the clocks to use UTC time (or when this is not possible, document the time zone that is used for components that waive this requirement)

Assumptions

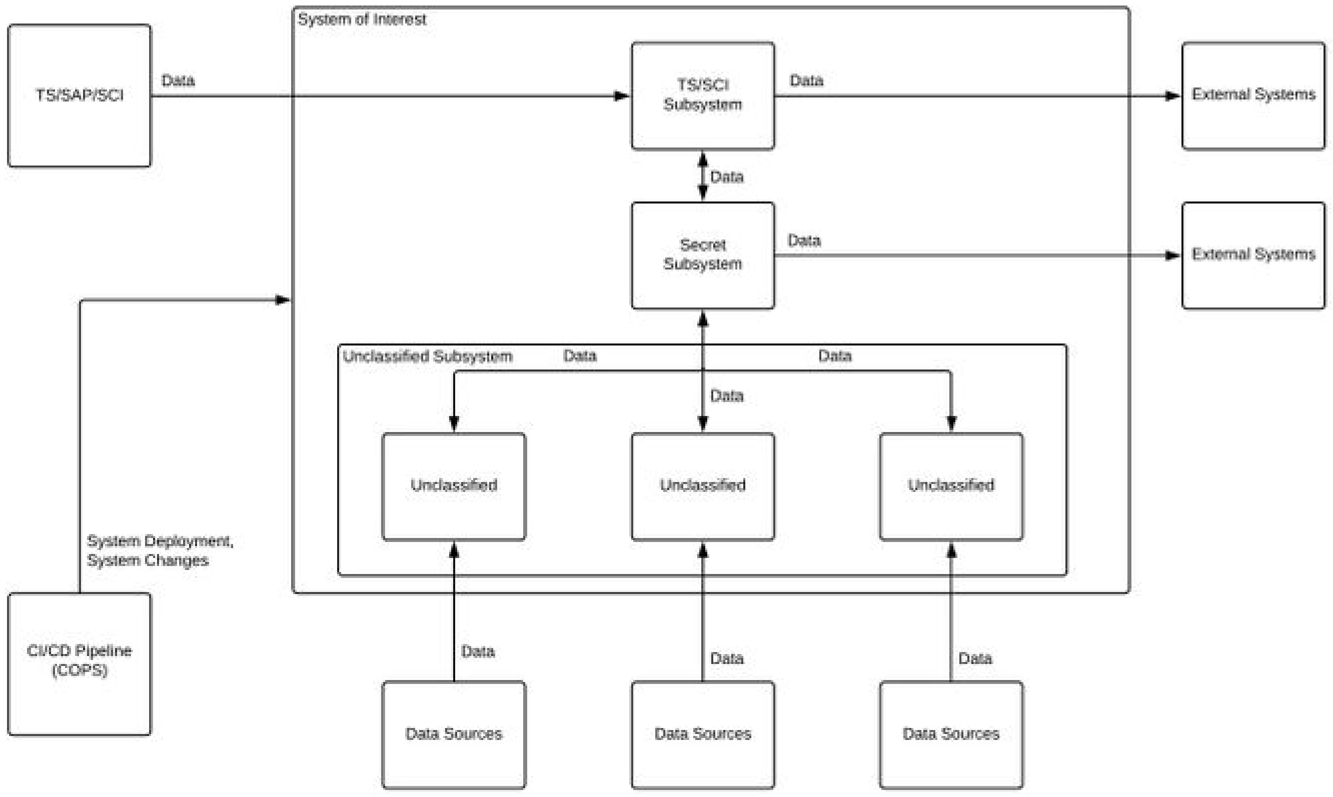
In the absence of clear requirements, prefer to design for flexibility. These assumptions constrain the scope of the design. If any become invalid, then the design will need to change, so please let us know!

- All hosted services provide a RESTful API to external consumers.
  - All data that passes through must be wrapped in one of the supported REST media types (usually JSON, XML, or HTML; but vendor-specific media types may be supported as well)
- The generic term 'data' is used throughout this document to refer to any series of bytes that can be transferred using REST. It is up to individual services to interpret that data properly.
- SELinux labels are the core of the Multi Level Security implementation. Every application that touches data must be aware of them (or at least pass them unchanged).
  - Include support for systems that use the Multi Category Security (MCS) instead of MLS.
- All services store data in a SQL database. In particular, a PostgreSQL database that uses SELinux labels.
- Humans provide the SELinux labels for all data. In future phases, a machine learning algorithm will make recommendations (requiring human approval). Eventually, that may evolve to radically reduce the human workload.
- The internal IdAM knows the level of the system where a user is requesting services (through something like a configuration file or information passed as part of the login credentials).
- All logs (included access logs, error logs, and status logs) are forwarded to an external log aggregator.
  - If log forwarding fails for some reason, an Alert must appear where an operator will see it (COPS-89).

Context

Figure 4 shows an overall architecture for the system where the COPS Platform resides. In particular, many of the arrows between classification levels would involve the COPS Platform, along with some internal messaging within each block.

Figure 4: High Level Architecture for COPS



References

- Overview of MLS Requirements and Guidelines
  - https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.2.0/com.ibm.zos.v2r2.e0ze100/ch1.htm
- Overview of RESTful APIs
  - https://en.wikipedia.org/wiki/Representational_state_transfer
- Documentation about SELinux labels related to Multi Level Security
  - https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/selinux_users_and_administrators_guide/mls
  - These are the best overviews I've found regarding the range field within an SELinux security context
    - https://selinuxproject.org/page/MLSStatements
    - https://wiki.postgresql.org/wiki/SEPostgreSQL_SELinux_Overview#MLS_and_MCS
- Here is a good summary of the difference between Authentication and Authorization
  - https://www.okta.com/identity-101/authentication-vs-authorization/
- Oracle's recommendations for a successful MLS implementation
  - https://www.oracle.com/database/technologies/security/label-security-multi-level.html
- This SE-PostgreSQL reference is from the initial slides that led to the creation of this COPS Platform document
  - https://selinuxproject.org/page/NB_SQL_9.3
- Amazon article about configuring a single API Gateway to serve different types of customers
  - https://aws.amazon.com/blogs/architecture/using-api-gateway-as-a-single-entry-point-for-web-applications-and-api-microservices/

Subpages

- Component: Platform Controller
- Component: ID and Access Management
- Component: Data Storage
- Component: Container Runtime
- Jira Stories (Related to COPS Platform)
- Research: MLS PostgreSQL

- Research: Docker SE Linux isolation
    - Securing Docker with SELinux
- Research: Firecracker Isolation
- Open Questions
- Data API for the COPS Platform

COPS Platform Components

This section provides some finer details of the capabilities of each component.

These should be taken into account when working on the implementation, and should be updated as things evolve and mature.

Figures 1 and 2 (in the Overview section) show how they relate to one another.

Identity and Access Management (IdAM)

- Authenticate users via an external IdAM system (such as a Directory Server) or with internal user/group configuration
- Retrieve Authorization information from an external IdAM system (such as a Directory server)
- Respond to queries about accesses for specific users
- Handle login across multiple applications
- Low-Level Design: Component: ID and Access Management

Platform Controller

- Authenticate users using an Identity and Access Management service
- Provide a single point of access to multiple RESTful services
- Provide a single point of access to Secure Data Storage
- Allow access to each service only to authorized users
- Secure (encrypt) data in-transit between consumers and producers of services
- Request containers for specific services for specific users
- Low-Level Design: Component: Platform Controller

Data Storage

- Provide a secure (encrypted) location for storing data at rest
- Provide a storage location for service data
- Structure data in a well-documented way
- Allow authorized users to access only their data
- Prevent users from accessing data without proper authorization (and log access attempts)
- Adhere to Multi-Level Security requirements and guidelines
- Authenticate users with the IdAM service
- Preferably use SELinux labeling to enforce access control
- Low-Level Design: Component: Data Storage

Container Runtime

- Provide compute resources for RESTful services behind the Platform Controller’s API Gateway
- Authenticate users with an IdAM service
- Ensure that only authorized users can access each container
- Start containers with specific access controls when requested by the Platform Controller
- Preferably use SELinux labeling to enforce access control
- Low-Level Design: Component: Container Runtime

Application selections

For each component, several options were considered. Details of the decisions are captured in Decision Logs (linked below) and conclusions are listed here.

Identity and Access Management

- Keycloak
- The project has been using Keycloak since the beginning.

Platform Controller

- Custom App based on Zuul 2
- Which API Gateway should COPS use?

Data Storage

- SE PostgreSQL
- Which MLS Database should COPS use?

Container Runtime

- Docker (with Kubernetes)
- Which container runtime are we using for COPS?

Interface Definitions

In the interest of designing each component separately, this section provides details of the interface between each component. Figure 5 shows the network ports that are used for communication between components. The arrows next to the port numbers point toward the server side of the connection.

Internal Interfaces

Identity and Access Management - Platform Controller

TCP Port 443 (IdAM as the server, Platform Controller as the client)

- User Credentials
  - Format: Probably openid or SAML
- Access Lists
  - Format: Probably openid or SAML

Identity and Access Management - Container Runtime

TCP Port 443 (IdAM as the server, Container Runtime as the client)

- User Credentials
  - Format: Probably openid or SAML
- Access Lists
  - Format: Probably openid or SAML

Identity and Access Management - Data Storage

TCP Port 443 (IdAM as the server, Data Storage as the client)

NOTE: This interface is currently unused.

- User Credentials
  - Format: Probably openid or SAML
- Access Lists
  - Format: Probably openid or SAML

Platform Controller - Container Runtime

TCP Port 2376 (Container Runtime as the server, Platform Controller as the client)

TCP Port 443 (Services inside the Container Runtime as the server, Platform Controller as the client)

- Container Request
  - Format: Probably: https://github.com/docker-java/docker-java
- REST messages to/from specific containers
  - Format is application specific

Platform Controller - Data Storage

TCP Port 5432 (Data Storage as the server, Platform Controller as the client)

- Data Request
  - Format: SQL
- Data Submission
  - Format: SQL

Container Runtime - Data Storage

TCP Port 5432 (Data Storage as the server, Services inside the Container Runtime as the client)

- Data Request
  - Format: SQL
- Data Submission
  - Format: SQL

External Interfaces

Identity and Access Management - External IdAM

Port/Protocol unknown

- User Credentials
  - ToDo: Format
- Access Lists
  - ToDo: Format

Platform Controller - Service Consumers

TCP Port 443 (Platform Controller as the server, Consumers as the client)

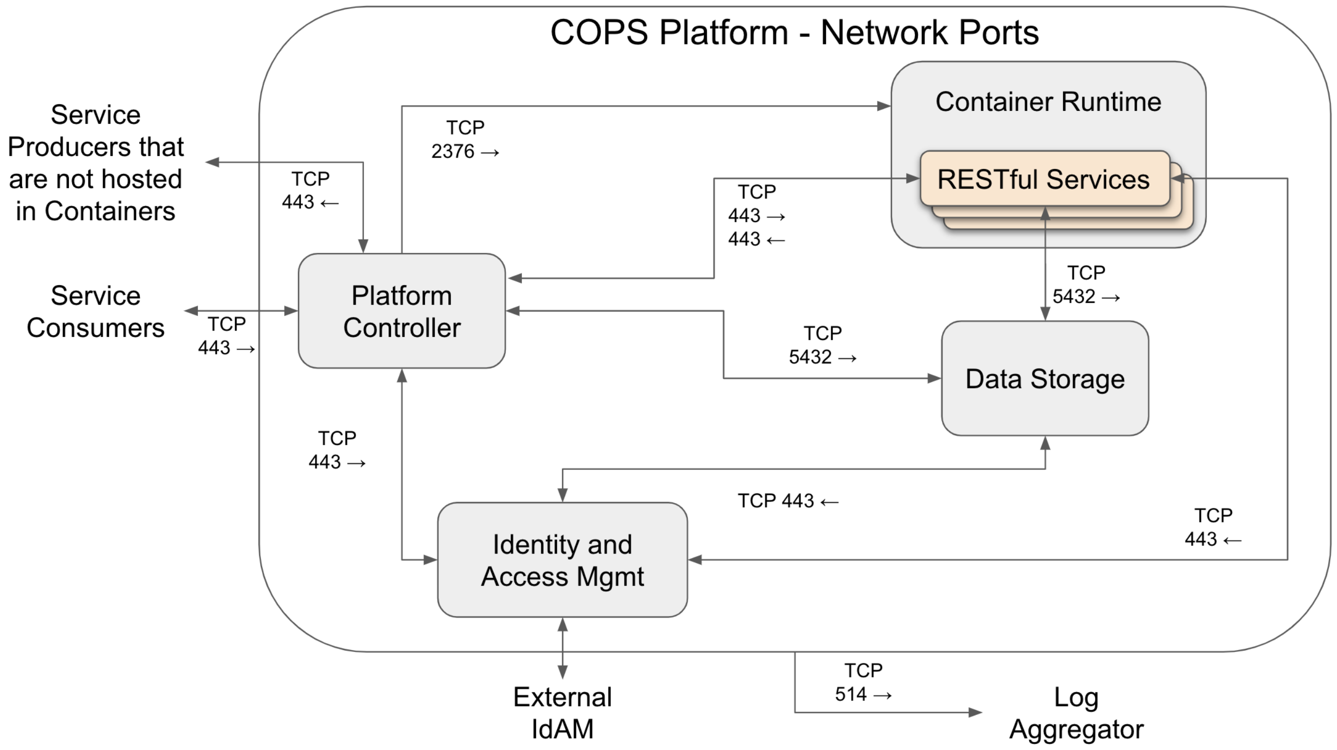
- REST requests to specific services
  - Format is application specific
  - Must include login credentials with the first request, and a token with future requests
- REST responses from specific containers
  - Format is application specific

All Internal Services - Log Aggregator

TCP Port 514 (Log Aggregator as the server, internal services as the client)

- Log file content
  - rsyslog (ToDo: Or do we already use a different standard?)

Figure 5: Network ports for communication between components



Security Notes

The other sections of this document contain many details about the security of the system. Here are some additional diagrams and notes to supplement that information.

Figure 6 provides a more detailed view of how the Container Runtime and Data Storage utilize SELinux to enforce isolation. Some of the finer details are still being researched (and may change).

Figure 7 captures the flow of user authorization information in a Communication Diagram, starting with a login event and continuing through all the components.

Additionally, the Activity Diagrams on this page provide more details of the flow involving SELinux: Data Flow Diagrams

Figure 6: Strategy for utilizing SELinux for data/compute isolation

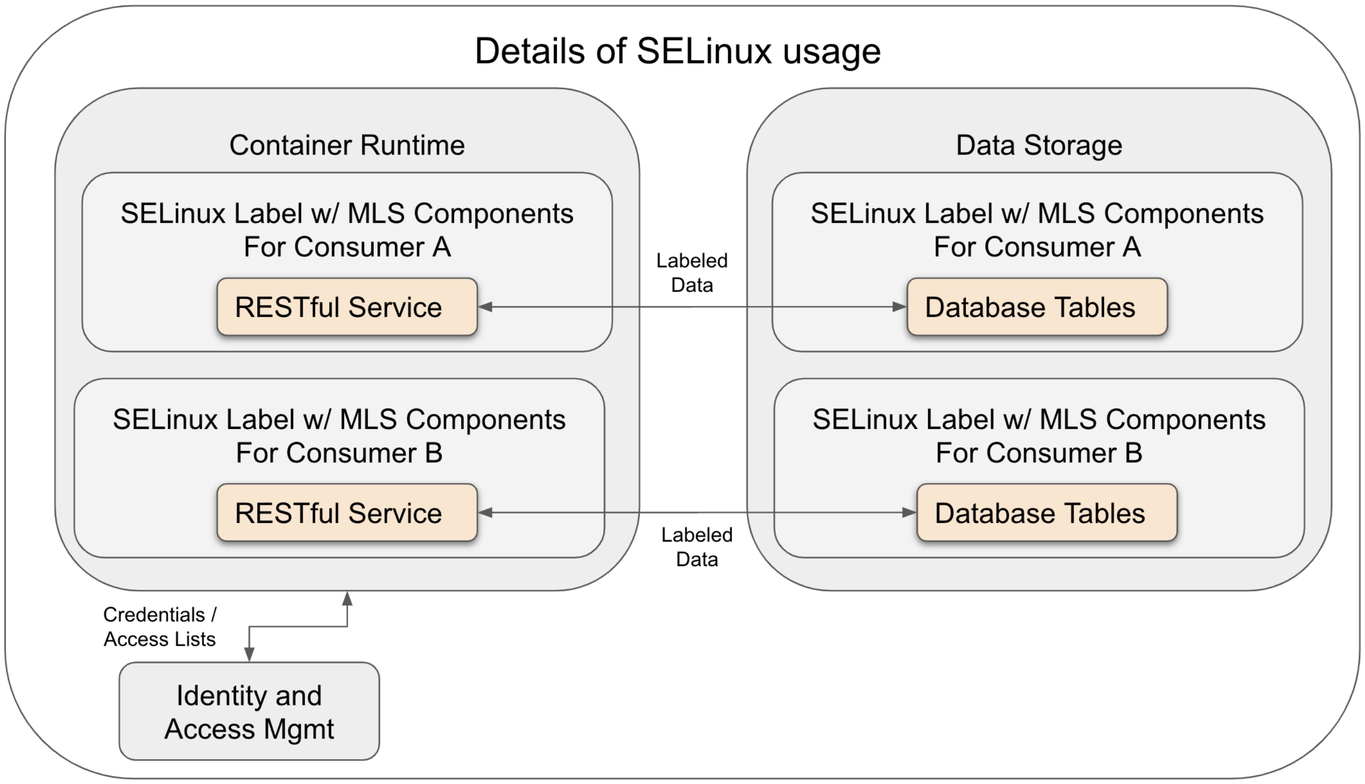
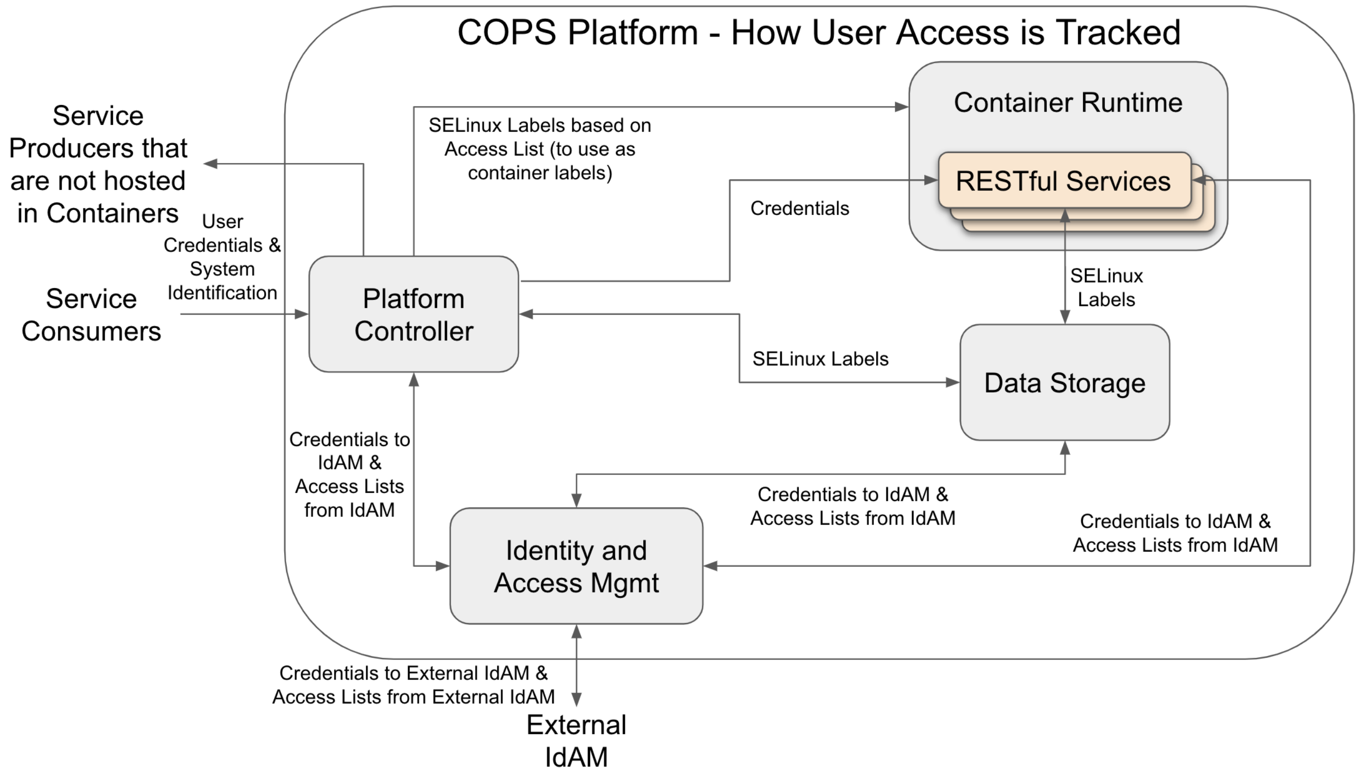

Details of SELinux usage
Container Runtime
SELinux Label w/ MLS Components For Consumer A
RESTful Service
SELinux Label w/ MLS Components For Consumer B
RESTful Service
Data Storage
SELinux Label w/ MLS Components For Consumer A
Database Tables
SELinux Label w/ MLS Components For Consumer B
Database Tables
Labeled Data
Labeled Data
Credentials / Access Lists
Identity and Access Mgmt

Figure 7: Passing user access information between components


COPS Platform - How User Access is Tracked
Service Producers that are not hosted in Containers
Service Consumers
User Credentials & System Identification
Platform Controller
SELinux Labels based on Access List (to use as container labels)
Container Runtime
RESTful Services
Credentials
SELinux Labels
SELinux Labels
Data Storage
Credentials to IdAM & Access Lists from IdAM
Credentials to IdAM & Access Lists from IdAM
Credentials to IdAM & Access Lists from IdAM
Identity and Access Mgmt
Credentials to External IdAM & Access Lists from External IdAM
External IdAM

Logging and Auditing

The low-level designs on the individual Components have some additional details, but here are some general capabilities and requirements.

- All logs must be forwarded to an External Log Aggregation server. They should not be kept locally (which would eventually fill up local

storage).
- All system clocks must be set to UTC time, and they must be synchronized with a central time server via NTP.
- Audit tools must be protected from changes by unauthorized users.
- Users must be automatically logged-out of sessions after a certain period of time. See the relevant STIGs or other documentation for specific timeout periods. This includes shell sessions, web application sessions, and any other user-initiated connections.

Sandbox Environment

There are several components in this subsystem that are configured independently. We have scope for some basic integration testing, so this section describes the Sandbox Environment where we can show basic functionality and sell-off some features.

3 Instances, all in a single private subnet

- One instance to host the Platform Controller
    - t2.medium (2 CPUs, 4 GB RAM)
    - 100 GB disk
- One instance to host the IdAM service
    - t2.medium (2 CPUs, 4 GB RAM)
    - 100 GB disk
- One instance to host the Container Runtime and Data Storage
    - m4.2xlarge (8 CPUs, 32 GB RAM)
    - 512 GB disk

Takes advantage of an existing public subnet with a Bastion and Reverse Proxy.

Reaching the Sandbox Environment

Currently, the Bastion is only accessible when connected to the Skyward IPSEC VPN. Additional IP address ranges could be added to the Security Group as-needed.

Bastion: 52.203.55.113

SSH Key in AWS: cops-dev (for the Bastion and the 3 instances mentioned above)