White Paper
Distributed Computing

Sentient Agent Bundle Resource Architecture

A system of systems solution for operating in hostile edge-to-cloud environments utilizing artificial intelligence to process, analyze and provide decision support on large volumes of data at the edge in the data center and the cloud

Authors
Darren W Pulsipher
Chief Solution Architect
Dr. Anna Scott
Chief Edge Architect
David Richard
Lead Solution Architect

Table of Contents

Introduction

As edge devices become more capable of processing data at the edge–including AI inference, data normalizations, encryption, and compression—organizations are looking at new operating models to drive decision-making closer to where data is generated and collected. This white paper aims to articulate the architecture and design of a distributed data stream framework utilized to process large volumes of data across a heterogeneous semi-connected ecosystem of edge devices named sentient agent bundle resources (SABR).

Problem Statement

Edge Computing brings new challenges that solutions in the data center or the cloud only do not have. Many organizations face edge ecosystems operating in a denied, degraded, intermittent, or limited (DDIL) network environment requiring systems that can work autonomously when network connectivity is in question. Problems with heterogeneous hardware and software platforms, large data generation and movement volume, and physical and cyber security make managing applications and devices at the edge complex. So complex that the industry is behind the projected adoption of edge computing.

Artificial intelligence (AI) is another technology that promises to open up new use cases and unlock data power to accelerate decision-making is artificial intelligence (AI). Moving AI to the edge accelerates decision-making by decreasing data movement volume and moving decision-making close to data generation. Because of the heterogeneity of edge computing, AI models are just as diverse as the types of edge devices. Organizations require a mechanism to manage heterogeneous AI models, data sources, and approaches in a unified manner across a distributed ecosystem.

Organizations need a solution that provides the following:

- Security – Provides a hardware root of trust to match AI models with attested hardware to prevent AI models from working outside the attested ecosystem. It also prevents unverified and unattested models from running in the ecosystem.
- Manageability – AI models and algorithms are managed from a federated control plane that contains deployment, updates, and decommissioning.
- Auditability – Changes in their AI models and algorithms are tracked from a single management framework that can be audited.
- Visibility – Enabled operator awareness of node health, AI model deployments, and operational efficiency.
- Reliability – Because the AI models are distributed across the ecosystem, there is no single point of failure; operational availability is sustained.
- Resiliency – The system must run in a DDIL environment.
- Consistency – AI model consistency is essential in a mission-critical distributed ecosystem. The approach must manage AI model divergence by keeping track

of AI model changes.

- Simplification – The framework provides resiliency, consistency, reliability, and security that all AI algorithms require under DDIL conditions during distributed operations. This simplification enables AI developers to focus on the AI application, not the complex environment.
- Scalability – System nodes can be added or removed easily without shutting down or reconfiguring the system. Streams and algorithms can also be easily scaled.

Design Pattern Concepts

The Sentient Agent Bundle Resource (SABR) provides an architecture that utilizes well-known concepts and design patterns to enable a resilient, easy-to-use architecture. These design patterns are the data stream pattern, container bundling in DevSecOps, container sidecar pattern, strategy pattern (policy management), zero trust architecture patterns, and reinforced collective learning pattern.

Data Stream Pattern

The data stream is a mature concept that allows data to be passed between a producer and a set of consumers without direct coupling between the entities. This concept provides the ability to deploy large numbers of producers and consumers in the same ecosystem without the fragility of a static coupled network. See Figure 1.

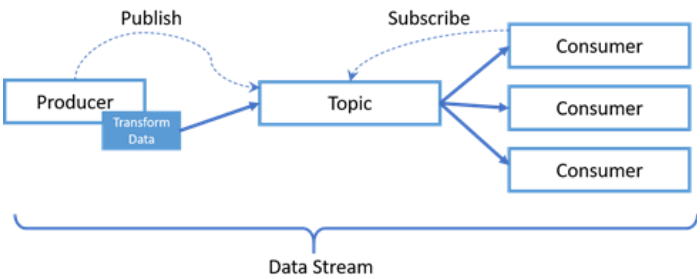

Figure 1: Data Stream Concept

A data stream is created when a producer (a service on a server) publishes data on a topic. One or more consumers subscribe to an issue and are notified when information is published on the topic. The producer, the topic, and the consumers create a data stream. Streams also contain a stream function mechanism to transform data before publication to the topic. These functions can also be utilized to identify event conditions and enable controls on stream management to subscribers for capabilities such as access, routing, event-driven alerts, and data prioritization.

Data streams benefit from loosely coupling the producer and consumer through an abstraction contained in the Pub-Sub framework. This enables the producer and consumer to operate independently of each other. This is beneficial during intermittent communication conditions since the consumer can continue to work even when not receiving data from the producer. Another benefit is that the producer can cache data if required and send it later. For example, if a device operates disconnected, the producer will cache the data and, when re-connected, publish the topic again.

Before a producer publishes the data, it uses a transformation algorithm to simplify or aggregate it. These transformation algorithms can perform functions including normalization, temporal compression, AI object detection, data aggregation, etc. These transformations are invaluable to getting the most compelling insight information to the right consumers (decision-makers) at the right time.

Managing these data streams, interactions, and transformation algorithms can only be accessed with a common framework. Open-source and commercial stream managers can manage the data streams across multiple locations, states of connectivity, and hardware platforms.

Benefits

This concept provides several critical benefits for the highly distributed, semi-connected environment.

- Dynamic heterogeneous mesh configurations.
- One or more consumers can subscribe to a topic fostering the reuse of familiar producers.
- A consumer can also choose only to consume data when a specific event occurs in a Data Stream.
- Resiliency when the network is down because a producer caches the data and, when re-connected, publishes data on the topic again.

Container Bundling in DevSecOps

Traditional DevSecOps pipeline pushes artifacts through a development, build, test, and deploy pipeline that produces an executable deployed into production environments. The container ecosystem has made the development of highly portable executables a reality by building container images that can contain several artifacts required to configure, secure, and execute microservices in any environment. See Figure 2 for an example CIDO pipeline using Data Streams.

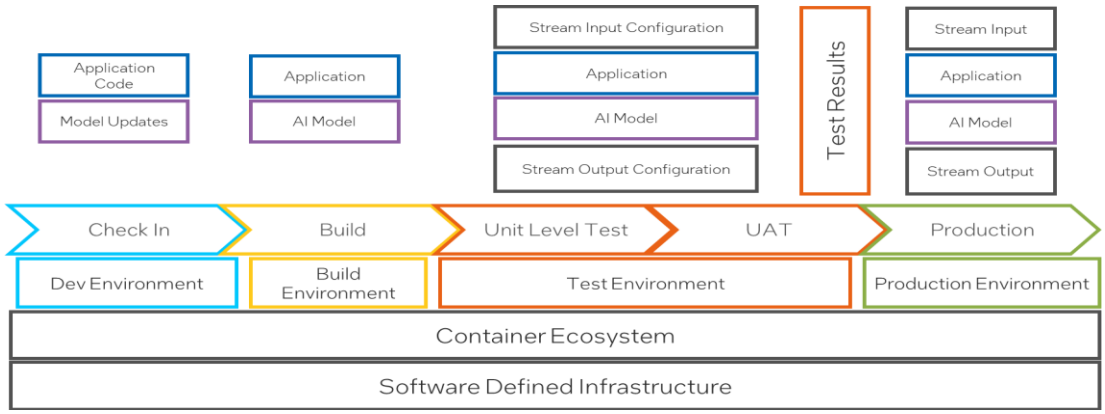

Figure 2: Bundling in DevSecOps

Traditionally Data Streams are configured in the production environment before applications are deployed. This pre-configuration requires coordination between application developers, system administrators, and program managers. They sometimes lead to longer deployment cycle times.

Leveraging the DevSecOps pipeline with container bundling means the container bunder contains the application, the AI Model or data transformation configurations, and the input & output stream configurations. In this scenario, data streams are configured at the deployment time of the container bundle, decreasing the potential problems with data stream configurations. Additionally, this bundling of all aspects of the data stream provides a mechanism to deploy producers and consumers of data streams anywhere in the edge ecosystem where containers can be deployed. The name of these data-transformation-centric applications is sentient agent bundles (SAB).

Benefits

This concept provides several vital benefits for deploying data streams across a distributed environment.

- Standard development and deployment of capabilities across a heterogeneous ecosystem.
- Portability of solutions from the data center into the edge devices.
- Decreased bandwidth to deploy only changes to bundles.
- Speed to deploy capabilities into the ecosystem.

Sentient Agent Bundle (Container Bundle)

The Sentient Agent Bundle (SAB) implements the container bundle pattern. It contains all the data required to deploy all microservices, stream configurations, and network configurations needed to run a data transformation algorithm (transformation, ai, analytics) on any node in the ecosystem and connect to the data streams for input and output. The pattern allows a bundle to be securely deployed on the edge, cloud, or data center. When a bundle has been verified and attested, it is unpackaged and deployed to the target node, launching all microservices and connecting them to the underlying stream management system. See Figure 3.

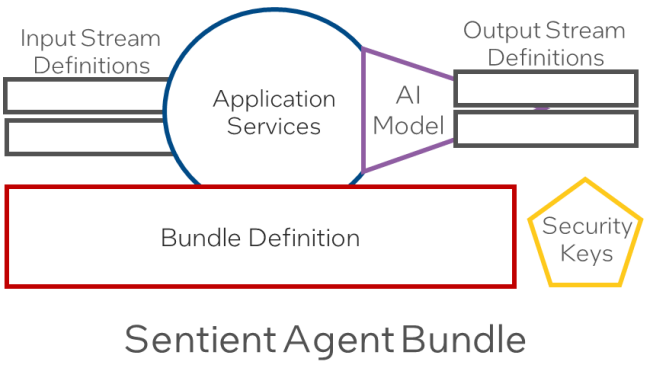

Figure 3: Sentient Agent Bundle

A SAB has the following characteristics:

- Sentient Agent Bundle (SAB), when deployed, manages transformation algorithms, data stream definitions, and interactions between systems.
- The container ecosystem (including Docker and Kubernetes) and DevOps environments (Red Hat OpenShift and Jenkins) build and distribute SABRs to Docker swarm and K8s clusters.
- The combination of all executables (applications and services), configuration files, stream definitions, data schemas, and transformation algorithms is called a Sentient Agent Bundle Definition (SABD).
- An SAB is represented as one container image in the Docker and K8s ecosystem and is deployed to a processor to bring it into the Learning Corpus mesh architecture.
- A security hash is added to the security keys in the Package and used to notarize the container image in a deployment repository.

Container Sidecar Pattern

A sidecar container is a helper container that helps manage administrative, logging, audit, or security tasks. The sidecar is typically a small container that does specific activities to support the main container. Many organizations use sidecar containers to add consistent behavior across heterogeneous containers like logging, audit, network encryption, and security.

Benefits

The primary benefits of the sidecar design pattern are:

- Management of multiple micro-services, data, and streams in a straightforward interface.
- Consistent configuration and support for strategy policy pattern.
- Consistent security, logging, and audit capabilities across all applications and micro-services.
- Elastic scalability of micro-services and resources based on telemetry from the service stack.
- Centralized control point for the application and data transformation.

Sentient Agent Bundle Resource Details (SABR)

While a Sentient Agent Bundle (SAB) contains the definition of a sentient agent, a Sentient Agent Bundle Resource (SABR) is a running instance of the sentient agent. It includes all the resources that enable the sentient agent to perform all the work designated for the agent. This includes evaluating the bundle against system policies for stream and channel definitions, establishing security domains, and managing learning-in and learning-out streams for each bundle.

When a SABR is deployed, a micro-service named the stream manager is created to manage the input and output streams of the application and services that transform information and publish on appropriate data streams. To connect to the correct streams, the stream manager evaluates stream creation policies deployed to the SABR platform or ecosystem. See Figure 4: Sentient Agent Bundle Resource

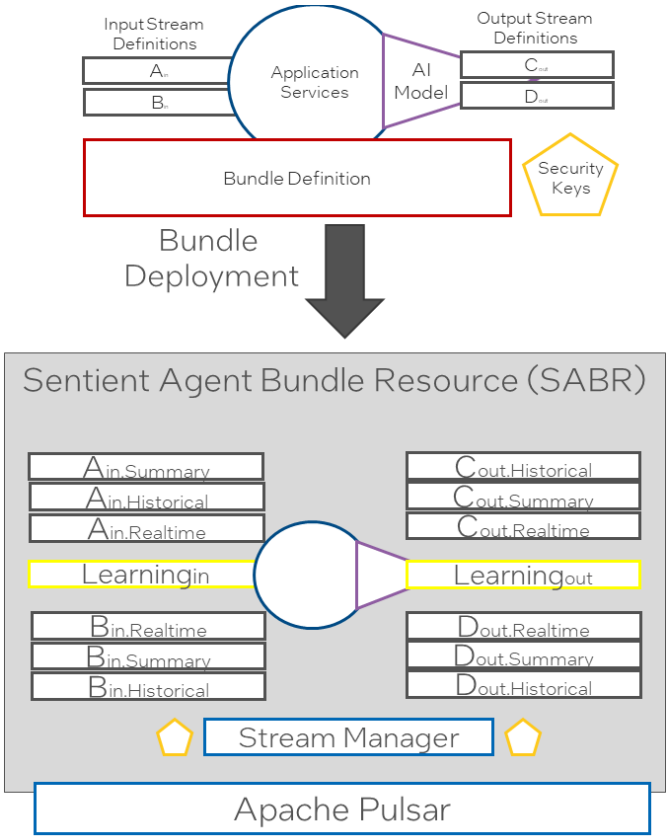

Figure 4: Sentient Agent Bundle Resource

To support a DDIL environment, each stream creates several channels that map to Data Stream Topics. Each channel is made based on the channel creation policies in the ecosystem, platform, or SABR definition. Channels can be turned on or off based on activation and prioritization policies for entities in the ecosystem. All communications in and out of the SABRs are through channels, not streams. A stream is an aggregation of channels based on channel creation policies. The following are the characteristics of stream management and channel creation.

- The Stream Manager creates the channels for each stream and monitors the streams based on stream creation policies
- A Learning Stream is created to connect to the learning corpus. Input and Output.
- For each stream definition, channels are created for each mode of operation based on the SABR, platform, or ecosystem creation policies. Example: Historical, Summary, Realtime
- Modes of operation are defined for all the applications, and the stream manager handles which channels to use during different modes of operation based on operational modes.
- Streams are encrypted and decrypted using security keys from the encrypted secret vault.

SABR Deployment (Sidecar Container Design)

A SABR container contains stream definitions, security keys, application definitions, AI models, and transformation algorithms. When a SABR is deployed, it explodes the configuration and deploys as many containers as needed to run the SABR. It configures the data stream manager to handle the streams and channels based on the system's policies. All communication to and from the application happens through the channels established in the stream manager.

Using the typical sidecar design pattern, the sidecar container unpacks all the elements for the sentient agent, including one or more data transformation microservices, data volumes that include AI Models, a stream manager micro-service, and any input and output channel adaptors required to convert stream data to be ingested by the intelligent agent services. Before deploying the microservices, the sidecar container validates the security certifications to run and decrypt data streams. See Figure 5.

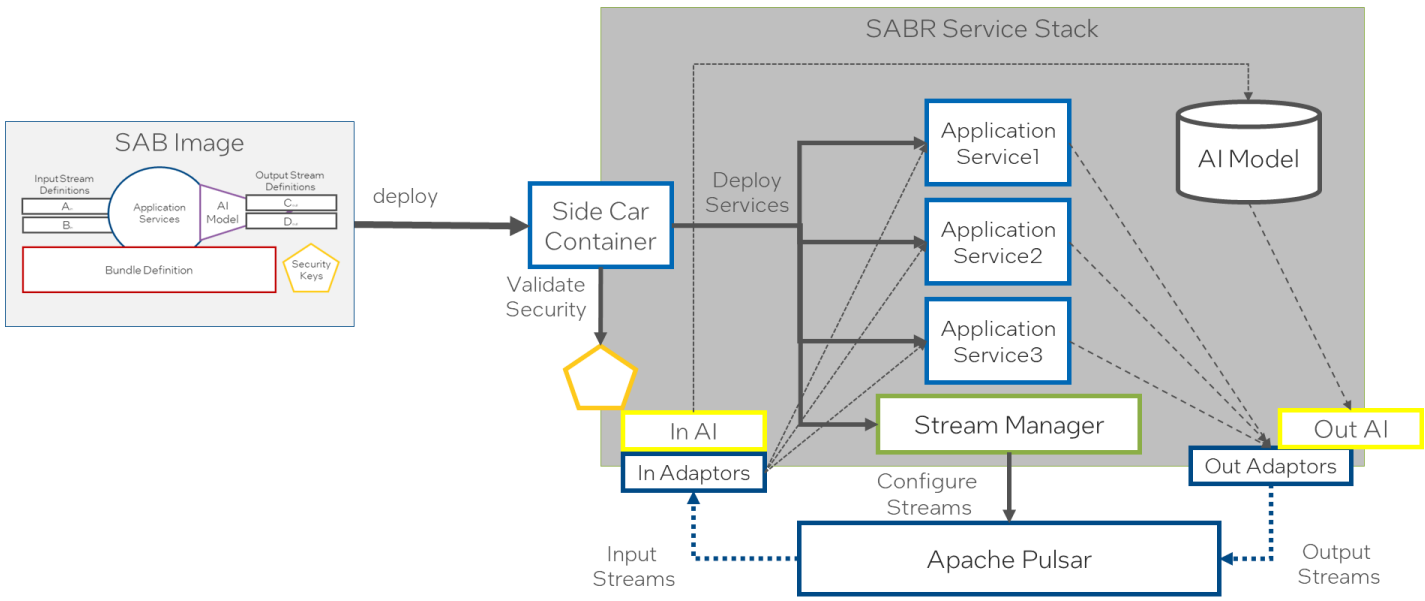

Figure 5: SABR Deployment

Strategy Pattern (Policy Management)

One of the critical requirements of the system is the ability to handle communications disruptions based on different detail environments. The ability to dynamically decrease the amount of data moving across the network requires using a pattern that allows for creating multiple streams based on environmental factors. The ability to dynamically recreate, activate, and prioritize communications flowing over the data streams is critical. For this reason, a strategy design pattern was selected.

A strategy design pattern allows applications to switch quickly between algorithms based on the runtime environment. The design pattern allows multiple behaviors to be defined without complex conditional if-then statements. The pattern extends behavior beyond the original architecture, providing more runtime decision-making. Modern architectures implement policy engines based on the strategy design pattern.

Benefits

The benefits of a strategy design pattern to implement a policy-driven architecture are:

- Dynamic definition of behavior at runtime.
- Centralized behavior management
- Simplified implementation of complex behavior
- Extendable such that it allows for new behaviors to be added to running application.

Stream Policy Management

The architecture leverages the strategy design pattern to build a stream and channel management policy engine. When a data stream is created, multiple channels handle the movement of data across the ecosystem. Each channel maps to a topic in the data stream design pattern. Channels are made based on policies defined for the ecosystem. The activation and prioritization of channels are also managed through policy. In the example in Figure 6, Stream A has three channels created that produce data at different frequencies and sizes. These other channels can be activated and deactivated based on the operating environment.

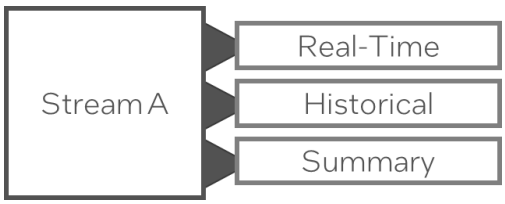

Figure 6: Stream to Channel Mapping

Two types of policies are leveraged in the architecture: creation policies and activation & prioritization policies.

Channel Creation Policies

One of the policies used in data stream management is the channel creation policy. When a data stream is deployed or a new creation policy is activated, a channel is created in the data stream ecosystem. A channel maps to a topic in a traditional PubSub system. Channel creation happens across the whole ecosystem, even if the channel creation policies apply to an individual SABR.

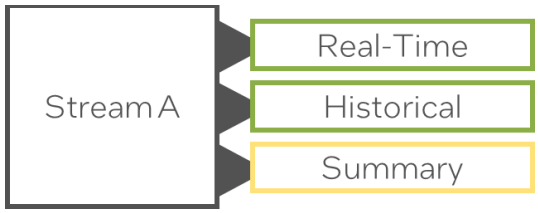

Figure 7: Channel Creation Policy

Channel Activation & Prioritization

The other type of channel policy is the activation & prioritization policy. This policy turns on or off channels based on policy and prioritizes the order of data transmission on the tracks when communications have been limited or operating at a lower bandwidth. The policies are dynamically deployed, and channels are activated and deactivated based on the policies. Figure 8 shows how activation policies are applied across all streams in the ecosystem.

Figure 8 Activation & Prioritization Policies

Policy Scope

Policies can be applied at different levels in the architecture. The policies are applied across the complete ecosystem, a platform, a group of SABRs, individual SABRs, or a particular device. A hierarchy of policies is evaluated from the tighter scope to the broader scope (Device -> Fleet). Figure 9 shows where policy can be applied in the ecosystem.

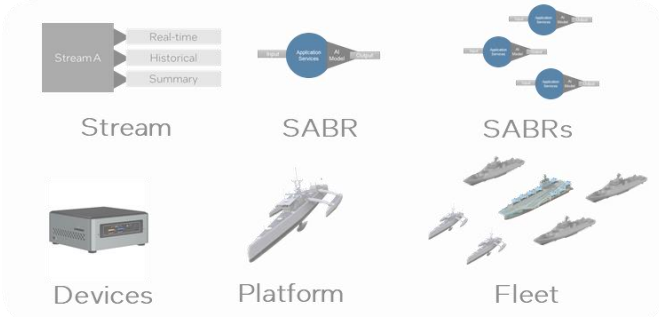

Figure 9 Policy Scope

Zero Trust Security

The "never trust, verify always" is the guiding principle of Zero Trust architectures (ZTA). One of the critical elements of verifying the elements in the ecosystem is to establish attestation, authentication, and authorization for all elements and their interactions in the ecosystem. ZTA requires temporal authentication that requires periodic re-authentication to ensure elements continue to be authorized to access resources in the ecosystem.

SABR Secrets Vault

To secure the graph of data streams, a security vault

containing all keys and hashes is included in the SAB to establish the root of trust between the SABR and the hardware it is running, the consumers of data streams, and the producers of data streams. See Figure 10.

Four areas must be protected to establish a zero-trust architecture of data streams.

- Prevent SABRs from running on untrusted edge devices. This prevents a bad actor from acquiring and running a SABR container on their hardware.
- Prevent untrusted/spoofed SABRs from running on trusted hardware. This prevents bad actors from deploying SABRs into a protected ecosystem, causing havoc, or stealing information.
- Prevent publishing of untrusted data onto a data stream. All data stream data is encrypted with appropriate shared encryption keys and hashes.
- Prevent receiving untrusted data from a data stream. Shared and private decryption keys and hashes are available to decrypt input data streams.
- Authorization and access to the data streams are time base and must be re-attested after the timeout.

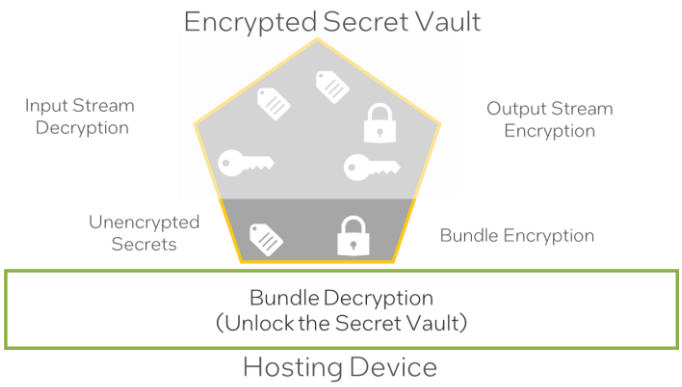

Figure 10: Encrypted Secret Vault

The security keys contain a secret encryption vault and an unencrypted security-critical section. When a SABR is deployed to edge hardware, the security keys in the unencrypted area are used to validate and decrypt the bundle, including the encrypted secret vault, which should be stored in protected memory, not persistently on the device. The keys and hashes stored in the encrypted secret vault decrypt and encrypt the input and output data streams.

Reinforced Collective Learning

In highly dynamic environments, AI models need to change. As AI nodes interact with the real world and are guided by human feedback, the AI models vary and adapt to produce better outcomes. As this new information becomes available, the ability to update AI models at the edge is critical. Sharing learnings through the propagation of model changes is essential to building a collective intelligent corpus that all edge nodes can leverage to perform their work. Propagating changes to AI models across both static and dynamic training of AI models is essential to heterogeneous platforms that include multiple edge and data center nodes.

Managing AI models in these environments is non-trivial and requires forethought and a robust system approach that includes DevSecOps. This is exacerbated by the working environment of edge deployments, where thousands of devices need to be updated with several dozen AI model updates in a continuous stream of updates.

Continuous Learning Stream

A systematic approach is required to manage the complexity of accepting, validating, and deploying dynamic and static AI model updates across the vast ecosystem. The SABR architecture provides the foundational elements to effectively manage AI algorithms at the edge.

The Learning Corpus is the intelligent, distributed repository of AI models. The Learning Corpus manages the AI models and their updates and tracks which SABRs are utilizing which AI model. These model updates are managed and validated in the Learning Corpus, then distributed to the SABRs in the ecosystem. This feedback loop is critical to controlling inconsistencies in AI models in the distributed ecosystem.

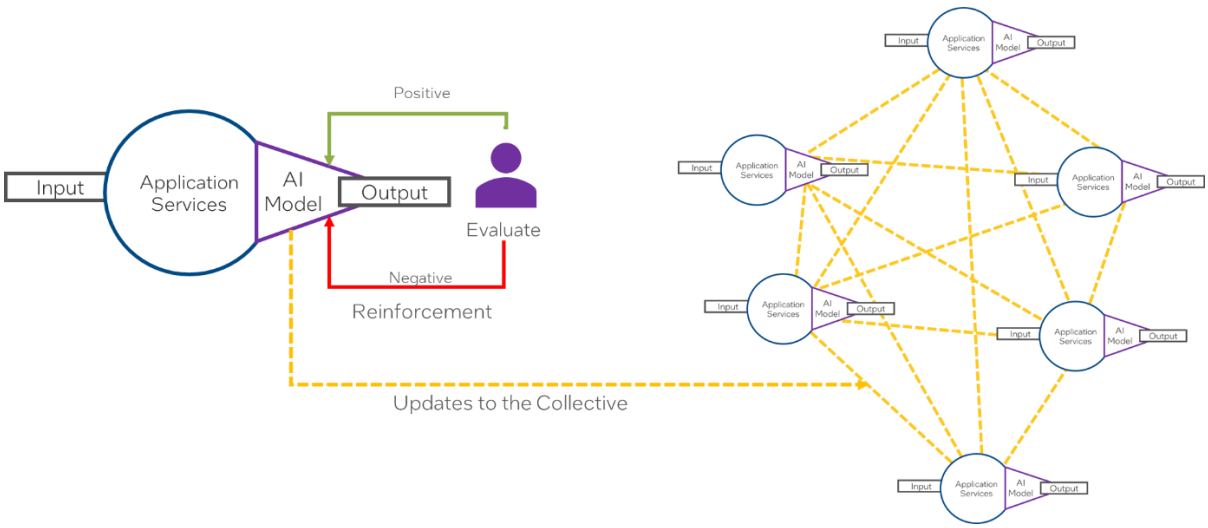

Figure 11: Reinforces Collective Learning

Solution

To enable a future-proof and expandable system, it is essential to understand how different parts of the system relate to each other and establish isolation layers (through standard interfaces or abstractions). This isolation allows the various subsystems in the solution to "grow" in parallel with minimal effect on each other. The SABR architecture should not be the only data management architecture utilized in the system. Leveraging common architectural elements is critical to developing a resilient and cost-effective approach. With the end goal in mind and the establishment of interfaces between the sub-systems, new features for hardware or software can be added progressively toward the utopian end state. This utopian architecture is known as Edgemere.

SABR Logical Architecture

The SABR architecture is an instantiation of the Edgemere Architecture and maps directly on top of the Edgemere Architecture. Not all elements of Edgemere are required for the SABR architecture. Assumptions are made that an SDI and Physical layer are already established in the solution. The following diagram, Figure 12, shows the subsystems specific to the SABR architecture.

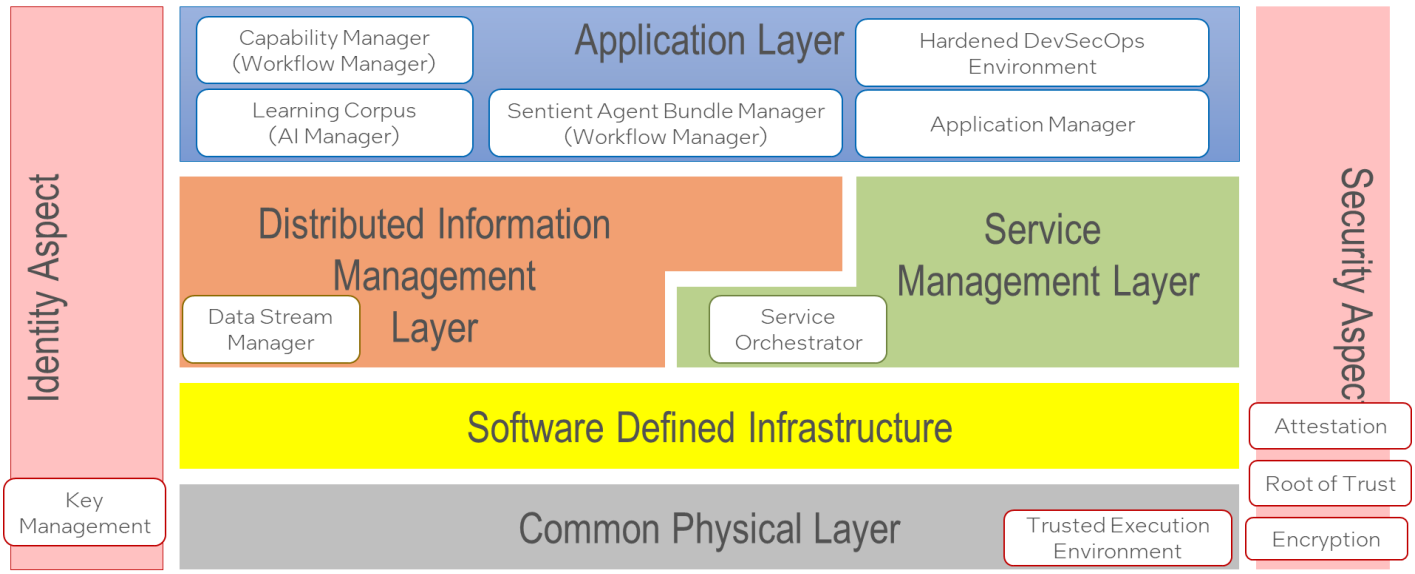

Figure 12 SABR Logical Architecture

- Application Manager – Responsible for the management (development, testing, and deployment) of applications in the solution.
- Capability Manager – Responsible for deploying and managing capabilities in the ecosystem, including the deployments of multiple SABRs.
- Data Stream Manager – Responsible for deploying, monitoring, and provisioning data streams in the ecosystem.
- Learning Corpus – Responsible for managing AI learning algorithms, updates, and deployments.
- Security Aspect – Gives a standard security model across the subsystems of the solution.
- Sentient Agent Bundle Manager – Gives the ability to bundle data streams, ai algorithms, and operate in a heterogeneous environment.
- Service Orchestrator – Responsible for deploying and managing services in the ecosystem.

Details about the SABR architecture can be found here.

Benefits

One of the benefits of this architecture is that applications can be developed in the data center and in a similar environment as what runs on the edge. User acceptance, unit level, and burn-in testing can all be performed on systems in the data center and then deployed in the field without variability in quality and operation. Because the Physical Layer is abstracted from the applications, the applications freely move between the different hardware. This easily lends itself to the portability of applications and capabilities and decreases the time to develop, test, and deploy applications and services. Putting a modern DevSecOps stack in the Application Layer can dramatically increase the deployment velocity. This also decreases the need for Digital Twin infrastructure and operations. Because the hardware is shared between the data centers and on-ship servers, building a complete digital twin is no longer needed. Only specialized hardware/application systems must be "mimicked" for digital twins.

Conclusion

The holistic approach to the system design has uncovered a complex problem space with a hostile environment that requires a highly resilient solution to handle heterogeneous devices, workloads, and services in a DDIL environment. The SABR architecture uses common design patterns in the security, DevOps, and application development space to improve the solution's scalability, reliability, and extendibility in this complex operating environment. Further details can be obtained by contacting Darren Pulsipher at darren.w.pulsipher@intel.com.