Adaptive Remote Gadget Unification System

Primary Owner opstechit-servicedelivery (LDAP)

Last modified 2 months ago by traykeli.

Adaptive Remote Gadget Unification System (ARGUS)

In Greek mythology, Argus Panoptes is a giant with 100 eyes, known as "all-seeing Argos". He is a good watchman because he never sleeps. Like Argus of old, ARGUS will be the watchman over all devices across OTS.

Glossary

List of terms/acronyms in relation to this document

Term	Definition
WAMS	Device management solution that leverages serial numbers access from 1st party sources as the unique identifier for devices
Deadbolt	Device management solution that leverages custom RFID tags as the unique identifier for devices.
4WAG	Telemetry data collected about a specific device, specifically who (i.e. userAlias), what (i.e. serial number, MAC address, etc), when (timestamp in milliseconds), where (in relation to this document WHID), action (TAKEN, RETURNED, LOST, BORROWED, etc.), and Gadget (WAMS, Deadbolt, etc).
Gadget	An OTS device which collects 4WAG telemetry data: WAMS, Deadbolt, etc.
ARGUS	Name of the system (Adaptive Remote Gadget Unification System)

Problem statement

Given that OTS has three different gadgets that track devices (WAMS 1.0, WAMS 2.0, and Deadbolt) each essentially gather the same core data (4WAG). There should be an alternative entity which aggregates and stores the data of the 4WAG collected by these gadgets which: enable high level visualization of all OTS products in one place, be in charge of device status change, sends alarms/ticketing of product issues, and acts as a data lake to make inferences on customer behavior.

The current solution to reconcile devices across different gadgets does not exist. A purposed solution was to use the WAMS 2.0 backend as the catch all for all future devices. This solution forces all future products into the paradigm set forth by WAMS 2.0's dependency on ADM device mapping. From a software standpoint, this leads to tight coupling. There is also a huge dependency on an external team, where we should be leveraging only specific features of said team (i.e. DeviceLock). Bringing that data to an in-house solution with well defined parameters (4WAG) enables a dedicated system who's sole responsibility is to reconcile devices, visualize live data, and to notify/fire alarms with full control and access of the data without unnecessary constraints. There is no way to do this today, nor was there any plans moving forward with this

- Relevant Links
  - Current MBR on the subject found here

Non functional requirements

This section describes the desired system characteristics for the proposed solution.

- Response time:
  - P99 <= 3000ms
  - Implementations of Lambda SNAP functions or increased provisioned concurrency
- Consistency guarantees
  - Strong consistency
- Request/Message volumes
  - Design plans on leveraging a connection to DynamoDB from AppSync, thus the default request quota's is well within the anticipated message volume
  - Product should adhere to message quotas found here
- Availability
  - Always available
- Durability of the data: TBD
- Security:
  - IAM roles that are either vended via an API to clients
  - IAM roles defined and shared for each Gadgets ecosystem (CDK package)
- Cost: TBD

Current system

There is no system in place to unify data or resolve the status of a device across different gadgets. In it's current state, Deadbolt devices can resolve, track, and display data surrounding devices when interacting within the Deadbolt ecosystem. I am making the assumption that the same is true for WAMS. Presented solutions were to integrate one existing system into another (Deadbolt into WAMS or vice versa). This does not scale and introduces added complexity to either system. As it stands, each gadget is already complex and dealing with the difficult problem of getting 4WAG data. That should be the only concern of any current gadget or future gadget that OTS intends to support.

Limitation of the present solution includes

- Integration of one gadget to another increase complexity and disregards Single Responsibility Principle (i.e. it is not WAMS/Deadbolts responsibility to manage the data collected outside of their ecosystem, it is their responsibility to gather and send it)
- Every future gadget will need to adhere to a given system (adding RFID stickers for deadbolt or ensuring that the value being passed to WAMS has an existing mapping in ADM). These constraints should be fulfilled and limited to the gadget abstraction layer
- If WAMS/Deadbolt goes down (i.e. the situation that happened via the move off corp) then the whole system goes down.

System context

The new system fits into the design because OTS-DSE is in its infancy in this initiative of device management across a series of products and choices made early on impacts future systems down the road. Decoupling this data unification layer from a given gadgets ecosystem not only enables our current gadgets to hone in on focusing on the proper collection 4WAG data, but this also introduces a standard onboarding process for future products with minimal overhead cost today.

Proposed solution

The Adaptive Remote Gadget Unification System (Argus) is the abstraction layer that provides a contract; given that a gadget publishes 4WAG data to Argus, the system ensures the following:

- A topic which can be subscribe to will stream 4WAG data as it comes in
- Ticketing, alarms, and high level device data visualization will be provided to clients that have been onboarded to the ARGUS platform
- API validation for device status (resolutions, last taken, etc) that can be manipulated and used at the gadget/client level

It will be the job of each Gadget to acquire accurate 4WAG data. Below is defined what ARGUS is expecting in terms of 4WAG data

Definition of 4WAG data:

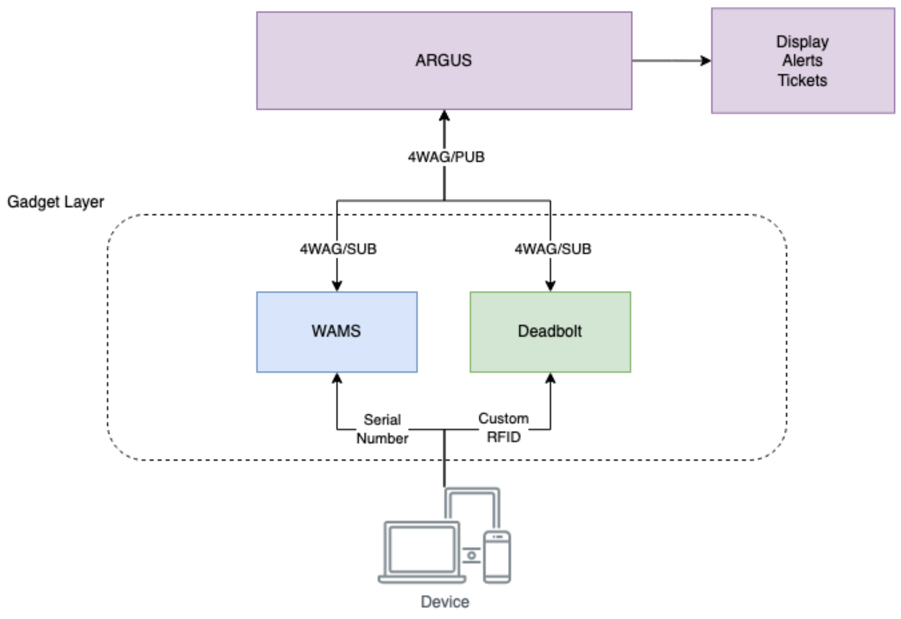
- who: `userAlias`
- what: `uniqueIdentifier + deviceType`
  - This value can be adjusted but for the sake of our product context and clarity of the document
    - TCX: serial number
    - MCX: serial number
    - RS: Custom RFID TAG
    - GLOVE: Custom RFID TAG
- when: `timestamp milliseconds`
- where: `WHID`
- action: `TAKE | RETURN | LOST | BROKEN`
  - These actions will be maintained by the ARGUS system and we can add/remove actions as needed
- gadget: `Deadbolt | WAMS`
  - Any future gadgets will need to be onboarded
- data: optional object data

Assumptions

Diagram's below make these assumptions

- Deadbolt has the ability to access serial number in it's own Gadget Layer
- WAMS has the ability to lock their screen via ADM API call (if necessary)
- Each device being used in a shared Gadget Layer will have appropriate hardware to perform calls
  - i.e. TC devices will have Custom RFID tag
- There is valid/stable internet connection to perform API calls
- Each device/gadget is properly provisioned

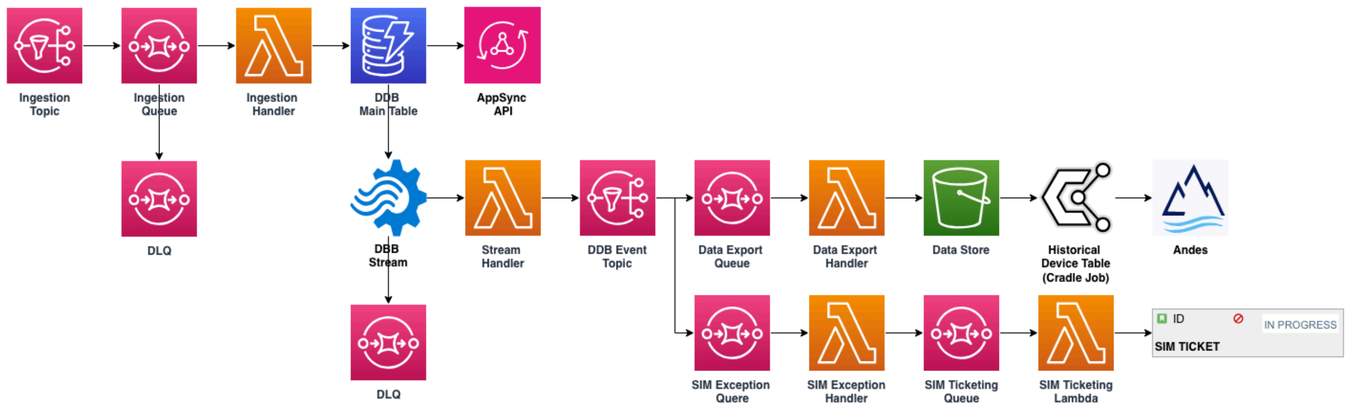
High Level Diagram



- Essentially, each Gadget will operate and work independently of each other only unifying via the 4WAG data on ARGUS. There will be an ARGUS ingestion topic which will receive gadget 4WAG data and there will be a subscription topic that each gadget can optionally subscribe to
- ARGUS will be in charge of Alerts (i.e. a device has been checked out for more than X amount of days, send an email to the WHO of the 4WAG data)
- ARGUS will be in charge of a high level dashboard
- For the sake of this document the concept of device resolution is any device that has a RETURN action (it is securely waiting to be used in a gadget)

Software Architecture

A breakdown of the different AWS components that make up ARGUS

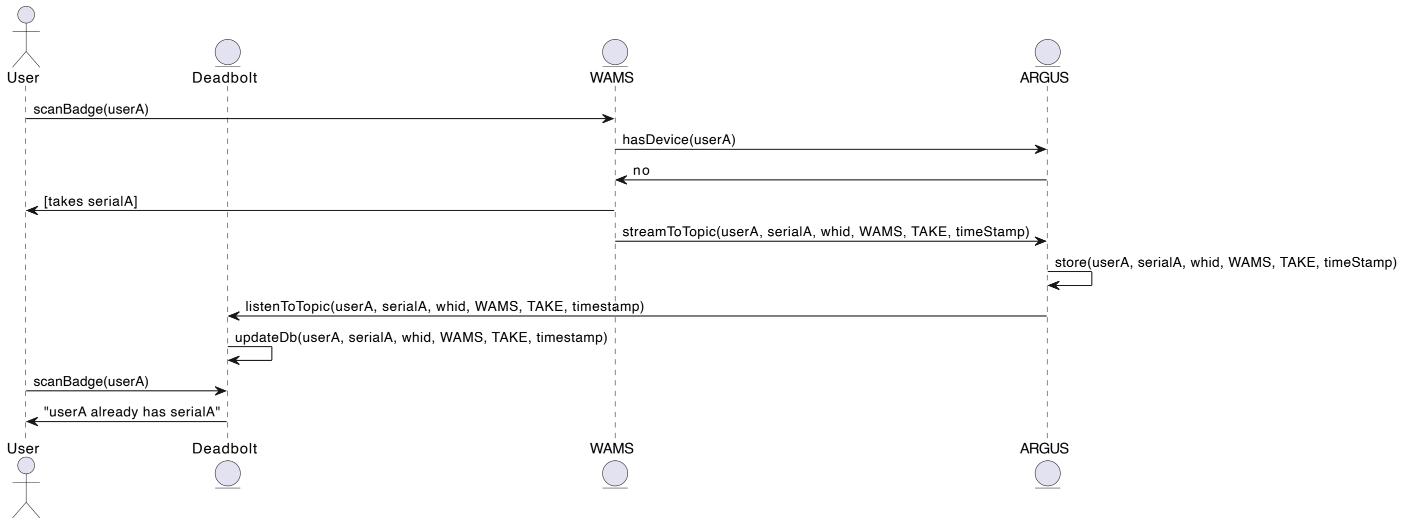


Design Inspector source below

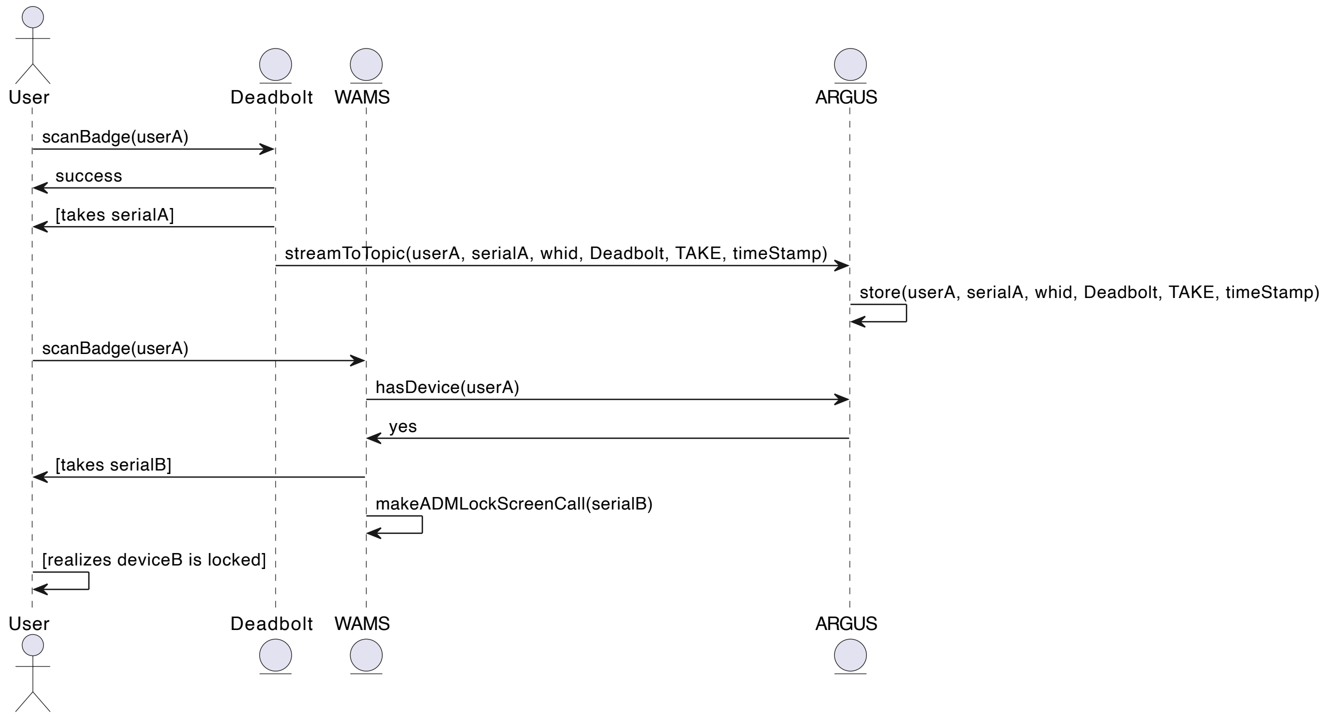
Show

High Level Sequence Flows

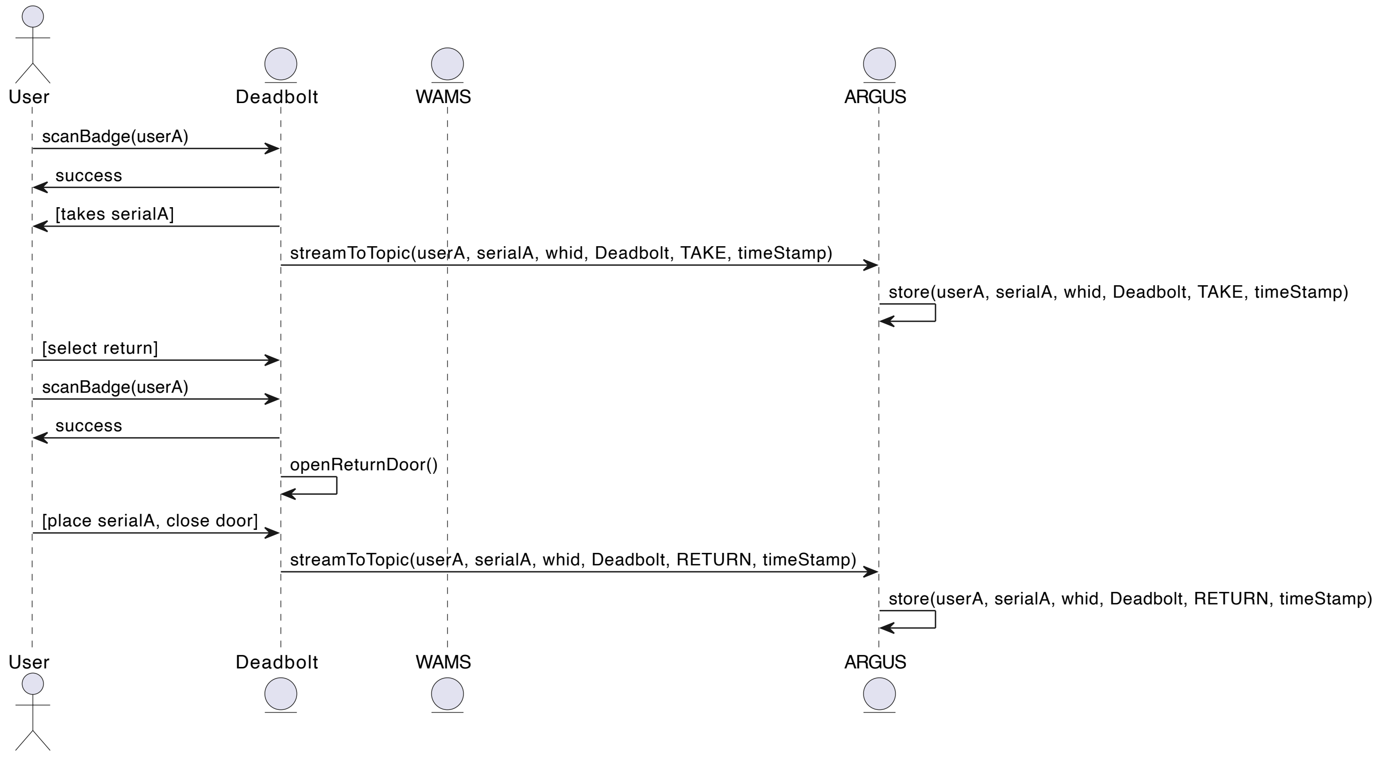
Scenario 1: Take from WAMS and Guardrailed by Deadbolt


User
Deadbolt
WAMS
ARGUS
scanBadge(userA)
hasDevice(userA)
no
[takes serialA]
streamToTopic(userA, serialA, whid, WAMS, TAKE, timeStamp)
store(userA, serialA, whid, WAMS, TAKE, timeStamp)
listenToTopic(userA, serialA, whid, WAMS, TAKE, timestamp)
updateDb(userA, serialA, whid, WAMS, TAKE, timestamp)
scanBadge(userA)
"userA already has serialA"

Scenario 2: Take from Deadbolt and Device locked by WAMS


User
Deadbolt
WAMS
ARGUS
scanBadge(userA)
success
[takes serialA]
streamToTopic(userA, serialA, whid, Deadbolt, TAKE, timeStamp)
store(userA, serialA, whid, Deadbolt, TAKE, timeStamp)
scanBadge(userA)
hasDevice(userA)
yes
[takes serialB]
makeADMLockScreenCall(serialB)
[realizes deviceB is locked]

Scenario 3: Take and return from Deadbolt


User
Deadbolt
WAMS
ARGUS
scanBadge(userA)
success
[takes serialA]
streamToTopic(userA, serialA, whid, Deadbolt, TAKE, timeStamp)
store(userA, serialA, whid, Deadbolt, TAKE, timeStamp)
[select return]
scanBadge(userA)
success
openReturnDoor()
[place serialA, close door]
streamToTopic(userA, serialA, whid, Deadbolt, RETURN, timeStamp)
store(userA, serialA, whid, Deadbolt, RETURN, timeStamp)
User
Deadbolt
WAMS
ARGUS

Data Storage

There will be two main Dynamo Document paradigms that will accomplish the following high level tasks:

- Real time Device Tracking
  - Live rows that get updated on the current 4WAG data of the devices (more on this below)
- Historical trail of Device Tracking
  - Basically the same thing as above, except the partition key will be unique (non-deterministic) so that the system can log all 4WAG events that have occurred

Note: <someVariable> will be replaced with parameters, otherwise the string literal shown below will be the assumed value.

Real Time Asset Tracking

	pk	sk	gsi1pk	gsi1sk	gsi2pk	gsi2sk	gsi3pk	gsi3sk
Schema	DEVICEID#<deviceId>	DEVICE	ACTION# <action>	USER#<userAlias># <timestamp>	WHID# <whid>	DEVICETYPE#<deviceType># <timestamp>	GADGET# <gadget>	DEVIC <times
Example	DEVICEID#00000011111	DEVICE	ACTION#TAKE	USER#traykeli#123456789	WHID#dtn6	DEVICETYPE#tc56#123456789	GADGET#wams	DEVIC

In this DDB paradigm a sparse index is used. When 4WAG data is inserted via the SNS topic the the corresponding deviceId passed is used to update the same row. This is due to the sk being deterministic (DEVICE). We should never have more rows of this type of data as we do devices Since it is deterministic (pk and sk are well defined and unchanging). This enables the system to validate whether some one has a device or not, when they last took a device, and where it was last scene. How that data is used belongs in the Gadget Abstraction Layer (i.e. deadbolt locking the doors or WAMS locking the screen). After X amount of days of a Row being untouched, an alarm can be sent out or a ticket can be made with the 4WAG data attached allowing customers to act accordingly.

This simple model also handles various edge cases (friends returning devices for others, hand-offs to managers after the shift, etc). All of this isn't bad actors and Argus aims to enable flexibility across different Gadgets and abstracts away processes that should be left on the Gadget layer. Consider the examples below and how device resolution is achieved by leveraging Argus:

Scenario A: User1 takes asset1 from WAMS, User1 returns asset1 to WAMS (happy path, 95% use case)

- Row initially had User1 4WAG data with a take action
- Row is updated with User1 4WAG data with a return action

Scenario B: User1 takes asset1 from WAMS, User2 returns asset1 to Deadbolt (on behalf of User1)

- Row initially had User1 4WAG data with a take action
- Row is updated with User2 4WAG data with a return action
  - On the Gadget layer each FC can do what they want for this behavior (discipline, nothing, etc). This should be the focus for these Gadgets, to cater to needs of the customer.

Scenario C: User1 takes asset1 from WAMS, loses it, goes to take asset2 from Deadbolt

- Row initially had User1 4WAG data with a take action
- Row persists with User1 4WAG data
  - This enables Deadbolt to act accordingly (lock User1 out of the system)

Scenario D: User1 takes asset1 from WAMS, gives to User2 to return, User2 loses it, User1 goes to take asset2

- Row initially had User1 4WAG data with a take action
- Row persists with User1 4WAG data
    - This enables WAMS to see that user has a device checked out and can lock the screen (also enables an opportunity for coaching, i.e. don't let others return your device, if that is how the FC operates)

Scenario E: User1 takes an asset, and never returns it *(negative happy path)*

- Row initially had User1 4WAG data with a take action
- Row persists with User1 4WAG data
    - consequence as needed per Gadget Layer

Historical device tracking

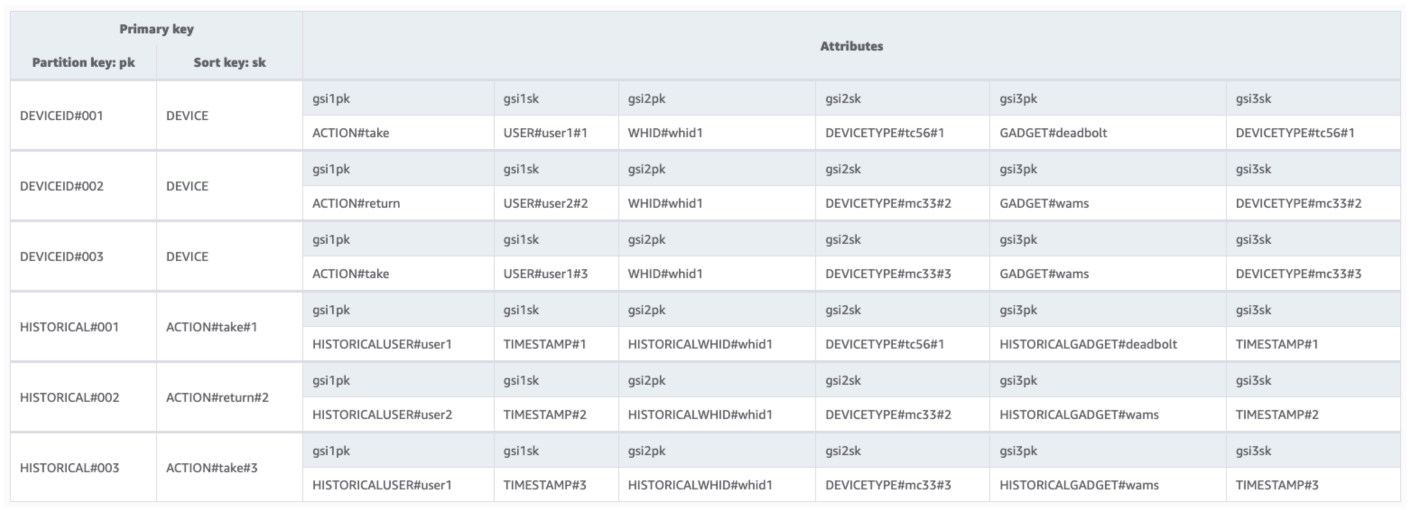
	pk	sk	gsi1pk	gsi1sk	gsi2pk	gsi2sk
Schema	HISTORICAL#<deviceId>	ACTION#<action>#<timestamp>	HISTORICALUSER#<userAlias>	TIMESTAMP#<timestamp>	HISTORICALWHID#<whid>	DEVICETYPE#<timestamp>
Historical Take	HISTORICAL#00000011111	ACTION#take#123456789	HISTORICALUSER#user1	TIMESTAMP#123456789	HISTORICALWHID#whid1	DEVICETYPE#

This document object tracks the 4WAG data obtained in the real time asset tracker. It acts as a log that Argus will send to Andes (in which big data inferences can be made over long periods of time) and short term historical events that can be exposed via an API, view examples below:

- See the complete history for a given device (get 4WAG data for the complete life cycle of any given device)
- See the complete history for a given userAlias (get 4WAG data for each device a user has interacted with)
- See the complete history for a given WHID in a date range

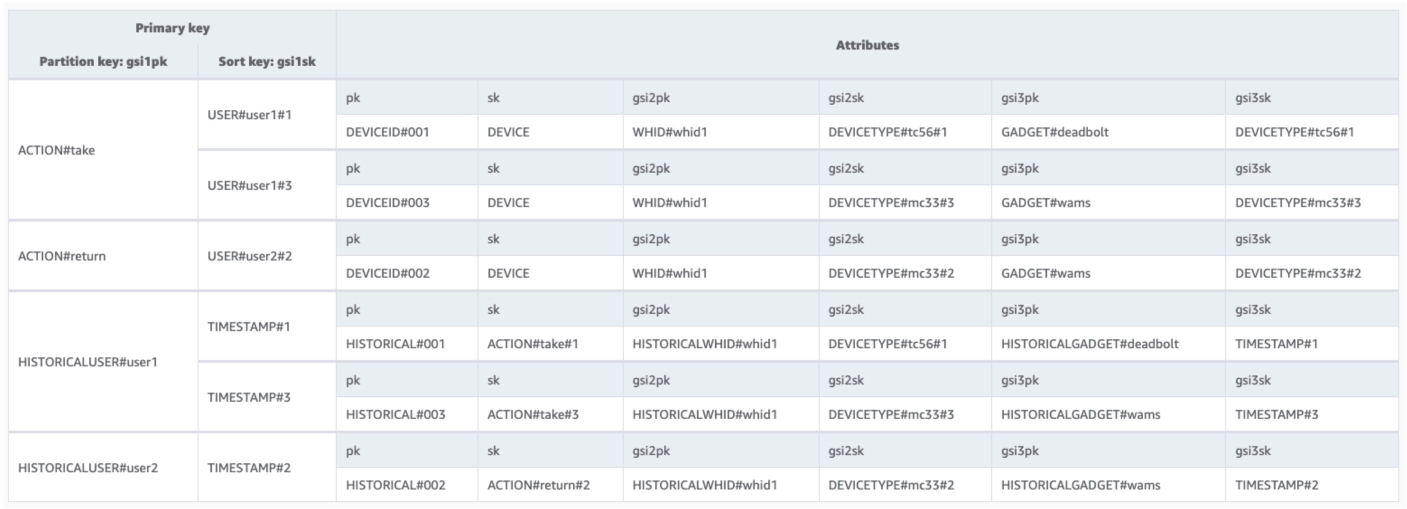
Dynamo DB Schema

Primary Key Query



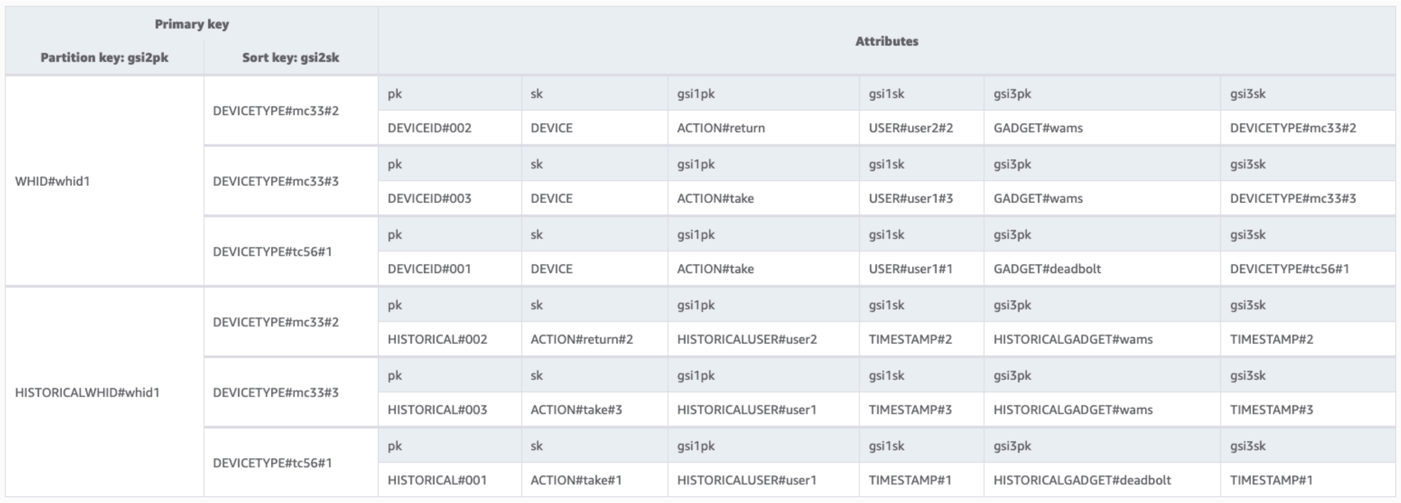
Primary key		Attributes					
Partition key: pk	Sort key: sk						
DEVICEID#001	DEVICE	gsi1pk	gsi1sk	gsi2pk	gsi2sk	gsi3pk	gsi3sk
		ACTION#take	USER#user1#1	WHID#whid1	DEVICETYPE#tc56#1	GADGET#deadbolt	DEVICETYPE#tc56#1
DEVICEID#002	DEVICE	gsi1pk	gsi1sk	gsi2pk	gsi2sk	gsi3pk	gsi3sk
		ACTION#return	USER#user2#2	WHID#whid1	DEVICETYPE#mc33#2	GADGET#wams	DEVICETYPE#mc33#2
DEVICEID#003	DEVICE	gsi1pk	gsi1sk	gsi2pk	gsi2sk	gsi3pk	gsi3sk
		ACTION#take	USER#user1#3	WHID#whid1	DEVICETYPE#mc33#3	GADGET#wams	DEVICETYPE#mc33#3
HISTORICAL#001	ACTION#take#1	gsi1pk	gsi1sk	gsi2pk	gsi2sk	gsi3pk	gsi3sk
		HISTORICALUSER#user1	TIMESTAMP#1	HISTORICALWHID#whid1	DEVICETYPE#tc56#1	HISTORICALGADGET#deadbolt	TIMESTAMP#1
HISTORICAL#002	ACTION#return#2	gsi1pk	gsi1sk	gsi2pk	gsi2sk	gsi3pk	gsi3sk
		HISTORICALUSER#user2	TIMESTAMP#2	HISTORICALWHID#whid1	DEVICETYPE#mc33#2	HISTORICALGADGET#wams	TIMESTAMP#2
HISTORICAL#003	ACTION#take#3	gsi1pk	gsi1sk	gsi2pk	gsi2sk	gsi3pk	gsi3sk
		HISTORICALUSER#user1	TIMESTAMP#3	HISTORICALWHID#whid1	DEVICETYPE#mc33#3	HISTORICALGADGET#wams	TIMESTAMP#3

GSI1 Query



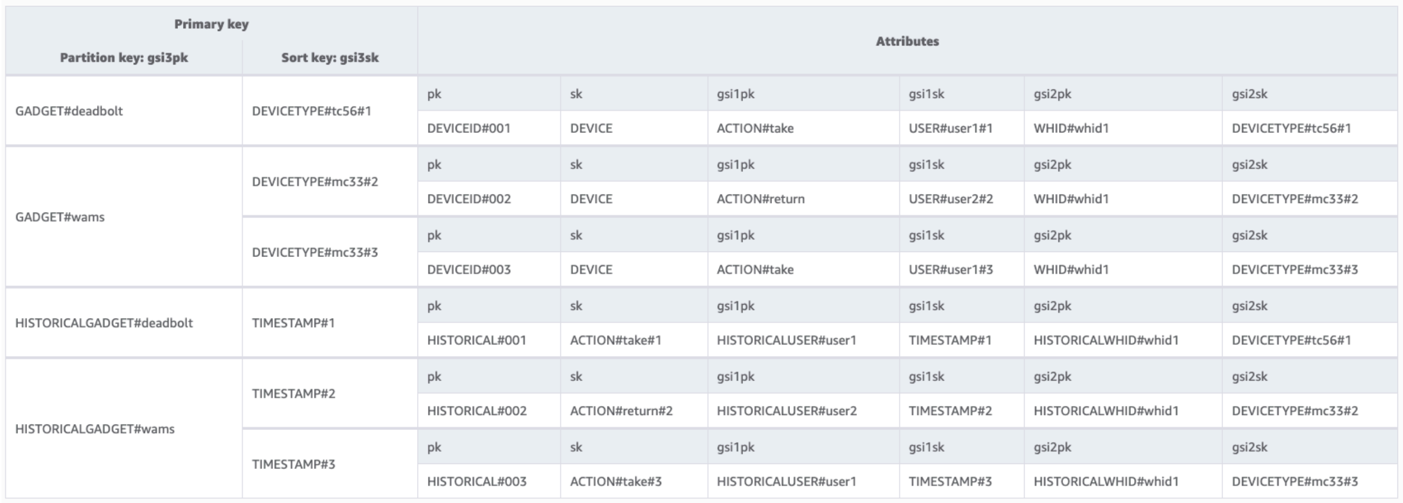
Primary key		Attributes					
Partition key: gsi1pk	Sort key: gsi1sk						
ACTION#take	USER#user1#1	pk	sk	gsi2pk	gsi2sk	gsi3pk	gsi3sk
		DEVICEID#001	DEVICE	WHID#whid1	DEVICETYPE#tc56#1	GADGET#deadbolt	DEVICETYPE#tc56#1
	USER#user1#3	pk	sk	gsi2pk	gsi2sk	gsi3pk	gsi3sk
		DEVICEID#003	DEVICE	WHID#whid1	DEVICETYPE#mc33#3	GADGET#wams	DEVICETYPE#mc33#3
ACTION#return	USER#user2#2	pk	sk	gsi2pk	gsi2sk	gsi3pk	gsi3sk
		DEVICEID#002	DEVICE	WHID#whid1	DEVICETYPE#mc33#2	GADGET#wams	DEVICETYPE#mc33#2
HISTORICALUSER#user1	TIMESTAMP#1	pk	sk	gsi2pk	gsi2sk	gsi3pk	gsi3sk
		HISTORICAL#001	ACTION#take#1	HISTORICALWHID#whid1	DEVICETYPE#tc56#1	HISTORICALGADGET#deadbolt	TIMESTAMP#1
	TIMESTAMP#3	pk	sk	gsi2pk	gsi2sk	gsi3pk	gsi3sk
		HISTORICAL#003	ACTION#take#3	HISTORICALWHID#whid1	DEVICETYPE#mc33#3	HISTORICALGADGET#wams	TIMESTAMP#3
HISTORICALUSER#user2	TIMESTAMP#2	pk	sk	gsi2pk	gsi2sk	gsi3pk	gsi3sk
		HISTORICAL#002	ACTION#return#2	HISTORICALWHID#whid1	DEVICETYPE#mc33#2	HISTORICALGADGET#wams	TIMESTAMP#2

GSI2 Query



Primary key		Attributes					
Partition key: gsi2pk	Sort key: gsi2sk						
WHID#whid1	DEVICETYPE#mc33#2	pk	sk	gsi1pk	gsi1sk	gsi3pk	gsi3sk
		DEVICEID#002	DEVICE	ACTION#return	USER#user2#2	GADGET#wams	DEVICETYPE#mc33#2
	DEVICETYPE#mc33#3	pk	sk	gsi1pk	gsi1sk	gsi3pk	gsi3sk
		DEVICEID#003	DEVICE	ACTION#take	USER#user1#3	GADGET#wams	DEVICETYPE#mc33#3
	DEVICETYPE#tc56#1	pk	sk	gsi1pk	gsi1sk	gsi3pk	gsi3sk
		DEVICEID#001	DEVICE	ACTION#take	USER#user1#1	GADGET#deadbolt	DEVICETYPE#tc56#1
HISTORICALWHID#whid1	DEVICETYPE#mc33#2	pk	sk	gsi1pk	gsi1sk	gsi3pk	gsi3sk
		HISTORICAL#002	ACTION#return#2	HISTORICALUSER#user2	TIMESTAMP#2	HISTORICALGADGET#wams	TIMESTAMP#2
	DEVICETYPE#mc33#3	pk	sk	gsi1pk	gsi1sk	gsi3pk	gsi3sk
		HISTORICAL#003	ACTION#take#3	HISTORICALUSER#user1	TIMESTAMP#3	HISTORICALGADGET#wams	TIMESTAMP#3
	DEVICETYPE#tc56#1	pk	sk	gsi1pk	gsi1sk	gsi3pk	gsi3sk
		HISTORICAL#001	ACTION#take#1	HISTORICALUSER#user1	TIMESTAMP#1	HISTORICALGADGET#deadbolt	TIMESTAMP#1

GSI 3



Primary key		Attributes					
Partition key: gsi3pk	Sort key: gsi3sk						
GADGET#deadbolt	DEVICETYPE#tc56#1	pk	sk	gsi1pk	gsi1sk	gsi2pk	gsi2sk
		DEVICEID#001	DEVICE	ACTION#take	USER#user1#1	WHID#whid1	DEVICETYPE#tc56#1
GADGET#wams	DEVICETYPE#mc33#2	pk	sk	gsi1pk	gsi1sk	gsi2pk	gsi2sk
		DEVICEID#002	DEVICE	ACTION#return	USER#user2#2	WHID#whid1	DEVICETYPE#mc33#2
	DEVICETYPE#mc33#3	pk	sk	gsi1pk	gsi1sk	gsi2pk	gsi2sk
		DEVICEID#003	DEVICE	ACTION#take	USER#user1#3	WHID#whid1	DEVICETYPE#mc33#3
HISTORICALGADGET#deadbolt	TIMESTAMP#1	pk	sk	gsi1pk	gsi1sk	gsi2pk	gsi2sk
		HISTORICAL#001	ACTION#take#1	HISTORICALUSER#user1	TIMESTAMP#1	HISTORICALWHID#whid1	DEVICETYPE#tc56#1
HISTORICALGADGET#wams	TIMESTAMP#2	pk	sk	gsi1pk	gsi1sk	gsi2pk	gsi2sk
		HISTORICAL#002	ACTION#return#2	HISTORICALUSER#user2	TIMESTAMP#2	HISTORICALWHID#whid1	DEVICETYPE#mc33#2
	TIMESTAMP#3	pk	sk	gsi1pk	gsi1sk	gsi2pk	gsi2sk
		HISTORICAL#003	ACTION#take#3	HISTORICALUSER#user1	TIMESTAMP#3	HISTORICALWHID#whid1	DEVICETYPE#mc33#3

Access Patterns

There will be two types of records to support the access patterns listed below :

- DEVICE
    - This record represents the live status of a given device (4WAG data that constantly updates the row)
    - The new user that checks out a device will then replace the data in this row
- HISTORICAL
    - Essentially duplicates the DEVICE row with a time stamp and keeps a record of 4WAG data with a timestamp as the SK

Gets

Access Pattern	Query Type	Required Parameters
Get DEVICE by action	gsi1 begins with query	action
Get DEVICE by whid in date range	gsi2 range query	whid, date range
Get taken DEVICE by user alias	gsi1 begins with query	user alias
Get returned DEVICE by user alias	gsi1 begins with query	user alias
Get DEVICE by device ID	pk sk query	device ID
Get DEVICE by gadget in date range	gsi3 range query	gadget, date range
Get HISTORICAL by user alias in date range	gsi3 begins with + range query	user alias, date range
Get HISTORICAL by device ID in date range	pk begins with + range query	device ID, date range
Get HISTORICAL by whid in date range	gsi2 begins with + range query	whid, date range

Creates

Create Type

Create Device Record
Create Historical Record

Updates

Update Type
Update Device Record
Update Historical Record

API

There will be an API, the implementation details falls a little bit out of the scope of the document, but in terms of the selection of GraphQL AppSync as the endpoint provider (according to the diagram) it is due to the power of subscriptions that an AppSync endpoint can provide. Essentially this (down the road) will entail that while the DDB table being updated in real time by each Gadget, a live stream of data will be exposed to AppSync which can be used to visualize data in realtime from a GraphQL endpoint. Dashboards, alerts, and other goodies will be made available from AppSync. This does leave the following open-ended until we fully understand the business need/if there is a use case for this presented design:

- What is the GraphQL Schema?
  - This should be trivial since the data should reflect 4WAG data
- What are all the Endpoints?

For context of this document an API will be available for the concept of `hasDevice` (as presented in the sequence diagrams above):

Request Body

```
{
  "userAlias": "traykeli",
}
```

Response Body

```
{
  "message": "user has a device",
  "success": true,
  "status": 200
}
```

While this may be trivial and seem unimportant initially, this will lay the foundation and enable expansion for an API suite which can be flexible and catered to business needs.

SNS Topics

Argus Ingestion Topic:

- This topic will be reserved for intaking 4WAG data
- There will be a unique ingestion topic per Argus environment (Alpha, Beta, Gamma, Prod)
- The lambda handler attached downstream will be responsible for insertion of 4WAG data in to DDB
- The value is TBD and will be generated from CDK at the environment level

Argus DDB Stream Topic:

- This topic will be streaming updates which have happened to the Table
- There will be a unique ingestion topic per Argus environment (Alpha, Beta, Gamma, Prod)
- Optionally, Gadgets may subscribe to this topic to receive updates as changes are made to the Argus data source
- The value is TBD and will be generated from CDK at the environment level

Alternate options considered

No other options were considered at the time of the creation of this design doc. During review we can visit and receive input from the team to explore future options

Technical debt

- During roll out there may be a time that we will be hard coding IAM role strings in config files manually for Deadbolt and WAMS in order to "on-board" new Gadgets
  - Once the process has been fleshed out and successful via manual intervention, a standard or automated route can be established for future Gadgets
- Lack of mapping of uniquely identifying individual gadgets (i.e. the difference between deadbolt locker usf2-1 vs usf2-2)
  - As it stands (from the doc standpoint) as long as you have the proper credentials and insert valid 4WAG data then Argus will update and manipulate the table accordingly
  - This is an open topic for debate since an argument could be made that all Argus cares about is the Gadget Family (Deadbolt vs WAMS) and not the actual Gadget ID itself (separation of concerns between Argus Layer and Gadget Layer)
  - The Dynamo DB single table design has all Strings and 3 GSI's which is highly flexible which can solve any direction the team decides to run with surrounding this topic
  - For P0 it would be recommended to add any unique data per individual Gadget (i.e. lockerId for deadbolt or MAC address for WAMS) in the optional data object (refer to definition of 4WAG data above)

- How to define the "what" of 4WAG and what this entails
  - There are data encodings at the Gadget layer which exists for Deadbolt (4 letter mappings which correspond to a given device)
    - should this data be standardized and adopted throughout Argus?
  - Agreeing on unique IDs that exist in industry standard has is what this document assumes and if those values are not easily accessible then using what is available from Gadget data
    - Serial number for TC/MC
    - Custom RFID from RS/Glove
  - When/If switching the what data definitions (maybe from serial to MAC address) there is no process in place to migrate that data called out in this doc
  - In short "what" data should never be changed (ideally) and should be agreed upon/readable by all current and future gadgets OTS plans to support. Argus itself will enforce that standard moving forward (imagine the situation we acquire another Deadbolt-esk product, they must adhere and be onboarded to Argus in order to leverage its benefits which includes being able to use/infer/detect "what" data agree upon by the system)
- On boarding of new devices (currently there are scanners) needs to be standardized
  - If OTS decides to use Argus to track Laptops, Badge Readers, etc. moving forward then there must be a process to on-board said devices
  - Lack of domain knowledge in this area led to the soft requirements in this field
  - Team needs to decide if Argus will be limited to Scanner tracking management or if it will be used at scale. If it will be used on a large scale basis testing with scanners and the development of a proper device on-boarding will need to be developed (not just conversations had in slack channels)

System Test-ability

- Integ Tests as an approval step in pipelines
  - This ensures that no bad code will be checked in
  - Also tests the different parts of the system as a whole
- Unit tests with 100% new line code coverage, 90% overall coverage
  - This ensures that each new function created works as expected
- Load/Stress testing
  - Along with Integ and unit tests ensuring that the amount of messages sent to a topic at once without failure will be crucial for Argus to grow at scale
  - It will be a good safety net to catch before pushing any code that may accidentally take down the system (non-obvious errors)
- Hardware Live tests
  - Adding a manual approval step and flow that ensures everything is working as expected in a given environment
    - Gamma stage (UAT) can be linked to a set of Gadgets in the AUS11 lab and only after a member of the team performs tests and validates things are working as expected is the code merge rolled out to prod using gradual deployment

Operational excellence

Top operational challenges with presented design

- **Creating and maintaining a repeatable process of boarding new Gadgets**
  - This can be mitigated with IAM roles, README.md, and wiki how to's initially (once a practice is solidified we can think about expanding streamlining this process via a webApp or some other mechanism)
  - This includes expansion of other project (potential gadgets) OTS acquires.
- **Establishing a process where Argus can alarm/ticket when a given Gadget hasn't responded in a given cadence and how to resolve/handle these cases**
  - This can be mitigated by various means, but this ultimately boils down to a health check for each Gadget:
    - creating a heartbeat detector model from IoT core
    - ping request/response model
    - scheduled API call
    - etc
    - This process will be left open for discussion amongst the team
- **Ensuring the data schema of the 4WAG model across all Gadgets**
  - This can be mitigated with Argus Smithy models which will have the interfaces Argus is expecting
  - Updates to said models will require proper roll out at scale, and notifying all Gadgets that they must update packages to new model before finalizing development
    - Consequences for such actions is currently ambiguous and open for debate (i.e. if the Gadget has not received a ping from Argus for X amount of time, how should the Gadget respond)

Important OE/On call tasks

- **Ensure that any Sev2 alarms are addressed**
  - **Purposed Sev2 issues**
    - Argus goes offline for X amount of time
      - How to mitigate: ensure that our deployment processes leverage proper tooling to prevent bad code from making its way into production
        - approval upon passing INTEG tests
        - step functions
        - gradual deployments (rolling back if there are more than X amount of errors over a period of time)
    - A given Gadget does not provide a health check update according to a defined cadence (i.e. Deadbolt at usf2-1 does not respond after 3 times in a 5 minute window)
      - How to mitigate: ensure we have well defined processes and runbooks for each gadget with resolution steps. Potentially locking down a given gadget if it has not received a ping in X amount of time

Launch requirements

Since Argus will act independent from the other systems, there are no stringent/hard and fast requirements surrounding launch. This section will be more catered to on-boarding of Gadgets to Argus and ensuring how that process will look. This will change if there is urgency in having both Deadbolt and WAMS in the same facility with the expectation of devices being resolved. The following sections are in context of the current business standpoint based on the Gadgets we are supporting in OTS right now. In theory, this application shouldn't affect the user experience of previous customers (a checkin/checkout of deadbolt/WAMS should feel the same as it has prior to Argus on-boarding).

Launch plan

The general launch plan formula for onboarding a new Gadget onto Argus is:

1. Integrate the IAM role into the Gadget's CDK package (or manually from AWS console if applicable)
2. Handle topic publish to Argus ingestion topic
3. (Optional) Handle subscribe stream from Argus DDB stream topic
4. (Optional) Integrate Argus `hasDevice` API into Gadget Layer

Deadbolt

- Ensure that serial numbers are extractable from current system (custom RFID tag mapping to serial number on the gadget layer)
- Integration of IAM role into CDK to allow streaming to Argus topic
- Integration of Lambda subscription topic to update Deadbolts main table
- Integration of Lambda publish topic to update Argus

WAMS

- Ensure that serial numbers are extractable from current system for supported devices (MC33 vs TC56)
- Integration of IAM role into CDK to allow streaming to Argus topic
- Integration of Lambda subscription topic to update WAMS
- Integration of Lambda publish topic to update Argus

Success metrics

- Number of Devices Unresolved
  - Definition of unresolved => device status of taken
  - If a device has a status of taken by the same user for more than X days it is considered Unresolved
  - An unresolved threshold numbers set per WHID will determine success
    - Green: X < 10 && X >=0
    - Yellow: X < 50 && X >= 10
    - Red: X >= 50
  - Note that these thresholds can be adjusted and explored either globally or per WHID basis, but for the sake of this document assume presented numbers at a global scale (so any WHID that reports these numbers will either be successful or non-compliant)
  - Emails can be sent to managers of useralias' that have any devices in an unresolved status
- To monitor system health, there should be a health check with each Gadget on-boarded to Argus
  - 5 minute timer from each Gadget to ensure secure connection
  - Consider the systems health compromised if X number of Gadgets missed Y health check calls
  - Further development and design around this topic can be explored at the feature level or in another design doc
- Number of open tickets
  - Consider Argus working as expected if WHID's that use the system reports less than X tickets weekly

FAQs

- It should be worth mentioning this doc was created without a WAMS SME and assumptions were made based off discussions and old documents
- This application assumes that there wouldn't be any personal information collected or distributed, thus security reviews for the application would/should be limited to useralias (further talk can be explored if needed regarding the use of personal data)
- Potential adjacent future projects that may stem from Argus entails:
  - On Board request portal
  - One dashboard (web app) for all device status
  - Custom trained LLM assistant for hardware instructions/resets per Gadget (reducing the cost/need of highly technical resources to trouble shoot issues)
  - Asset management outside of scanners (potentially streaming ITHUBX data and viewing ITHUBX as a gadget, unifying assets as a whole)
  - Broken device financial planning
    - Keep a total of devices that have BROKEN status
    - Ensure that the amount of devices requested match up to the amount of BROKEN devices per site
    - Ensures/keeps the integrity of devices per WHID (saving costs)

Open questions

1. How should we go about onboarding devices (e.g. creating support for glove scanners for instance)
   1. How do we determine the WHAT in 4WAG (serial number vs MAC vs RFID tag)
   2. Where do we keep this mapping?
      1. Could be stored on the Argus single table design?
      2. Could be stored in a config file?
      3. Could be created from a custom onboarding web portal?
2. If we change any 4WAG data methodology (switching to RFID number instead of useralias for instance), how do we migrate/handle existing data?
3. How do we address the situation where one Gadget is not connected to Argus, but another is working fine (i.ie Take from WAMS and the serial number does stream to Argus, the user would then be able to take another device from deadbolt)

High Level Tech Scoping

The scope breakdown below uses the following t-shirt sizing to estimate the work required:

- XS - 1 dev week
- S - 2 dev weeks

- M - 4 dev weeks
- L - 8 dev weeks
- XL - 16 dev weeks

Each work item assumes testing (unit/integration testing) in the estimate (where applicable)

Work Item Number	Service/Component	Title	Description	Estimate
1	Argus	Argus Scaffolding	Create the CDK, Service, Integ test, and Model code packages scaffolding for alpha, beta, and prod accounts for Argus	S
2	Argus	Argus Main Table	Creation of DynamoDB table with proper PK, SK, and GSI schemas as outlined in the doc	XS
3	Argus	Argus Data Ingestion	Create a queue + lambda that receives 4WAG data messages from the Argus Ingestion topic and upserts the data into Main Table	S
4	Argus	Argus DynamoDB Stream	Create a Dynamo DB stream lambda that sends 4WAG data upon table update to the Argus DynamoDb Topic	S
5	Argus	Argus Data Export	Update Argus CDK code and service to support data export. Create cradle job and make necessary internal requests to get data into Andes	M
6	Argus	Argus SIM Exception Ingestion	Create a queue + lambda that receives DDB exception messages from the integration topic and inserts them to the SIM ticketing queue	S
7	Argus	Argus SIM Ticketing Integration	Create a queue + lambda that receives exception messages from Argus SIM Exception and send the appropriate data to the Tickety API	S
8	Argus	Argus AppSync	Update Argus CDK to support AppSync API and GraphQL schema. Integrate API lambda call into Argus Service package.	XS
9	Argus	hasDevice API	Create a hasDevice endpoint in AppSync which given a useralias returns whether that user has a device or not	XS
10	Deadbolt	Convert TAG to Serial	Update DDB Paradigm to store Serial numbers (encoded on the RFID tags), ensure that the serial number stored on Deadbolt DDB matches the PK/SK values stored in Argus	S
11	Deadbolt	Argus DynamoDB Stream Subscription	Update deadbolt CDK to ensure subscription IAM roles and permissions are enabled to Argus DDB Stream. Create a lambda which handles the streamed data and updates the corresponding DDB record	M
12	Deadbolt	Argus Data Ingestion Publish	Update deadbolt CDK to ensure publish IAM roles and permissions are enabled to Argus Data Ingestoin. Create a lambda which handles the Deadbolt Data and transforms it to 4WAG data to be ingested by Argus	M
13	WAMS	Integrate Argus	I will leave this section open-ended being that I do not have a full picture regarding the implementation details around WAMS. I assume the same integration details for deadbolt applies here, but during design review WAMS SME's can fill out requirements here if needed	?

Tags: