Civic Architect Challenge - Kevin

Disclaimer

- For my solution I use Lambdas + DynamoDB for their autoscaling capabilities. For the scope of this assessment I felt those were the obvious choices. In the real world I would more carefully consider strengths and weaknesses of Lambda vs EC2, RDS vs DynamoDB, etc. I would implement load balancers where needed, apply database sharding, use proxies, etc. Instead, I kept it simple so you can focus on the MVP.
- I don’t implement any request body schema validation in my solution for simplicity.
- I don’t apply caching throughout my solution, again for simplicity.
- Since the solution must ideally be event/message based I consider AWS Step Functions or Lambda chaining non-viable options.

Solution

I think the best approach for system like this is to compose it out of loosely-coupled services. These services should all prioritize communicating via *EventBridge* through an EventBus. Using this approach we minimize dependencies between services. The only real dependency is EventBridge and the PII data format. However, if the latter is respected throughout release cycles, one could argue there are no dependencies.

This approach has a couple of other benefits:

1. It is easier to maintain/update services. The only dependency these services share is that they need to be able to communicate via *EventBridge* and the PII data format should be respected by all services involved.
2. It allows for teams to take ownership of specific domains, thus enabling specialization.

3. Each of the services can be deployed independent of each other thereby enabling:
    a. Scaling.
    b. The ability to swap out old services for newer ones without disrupting the entire system which minimizes down-time.
    c. Easier planning and deciding upon release schedules.

Scalability

I've specifically chosen to use AWS Lambda and DynamoDB in my solution because of their auto-scaling capabilities. I think it's hugely beneficial to the velocity of a team if scaling is not a concern. AWS does this for us afaik. However, if we would run into scaling issue in the future my approach would go something like:

- Add a load balancer that routes requests to multiple instances of the same lambda functions.
- Add replica sets to our database solution, again to balance the load.

Databases coupling

Even though microservices solve a lot of problems they also introduce some difficulty when it comes to interacting with databases. Let's say we have to lambda functions that depend on the same data. We could have them connect to and use the same database but this breaks the *decoupling* principle.

Ideally we want a microservice to be completely isolated, meaning that each services has its own database in order for the service as a whole to be independently deployable. In practise however this can be very difficult to achieve because the data needs to be replicated between the database instances somehow. There are solutions for this problem however

One such solution is using Sagas whereby each services independently updates its local database based on a transaction type in sequence. However I felt that for this project such a mechanism was out of scope. My approach instead is to have shared databases but designing the overall system in such a way that:

1. Minimizes reads and writes.
2. Reads happen on one lambda functions while writes happen on the other.

3. Always use ACID transactions whenever possible.

Components

Name	Type	Responsibilities
API Gateway	API Gateway	Responsible exposing API methods via public url.
EventBridge	EventBridge	Responsible for receiving and emitting events.
KMS	KMS	Responsible for encrypting/decrypting PII.
CivicGateway	Lambda	Responsible for encrypting PII. Responsible for generating UUIDs. Responsible for reading decision data from low security datastore.
CivicValidator	Lambda	Responsible for validating PIII validation requests. Responsible for writing decision data to low security datastore.
CivicPIIStorage	Lambda	Responsible for reading/writing PII to/from high security datastore.
CivicAnalytics	Lambda	Responsible for analytics pipeline.
HighSecDataStore	DynamoDB	Responsible for storing encrypted PII.
LowSecDataStore	DynamoDB	Responsible for storing unencrypted decision data.

Events

Event name	Triggered when	Payload
ValidationRequest	User sends POST request to *CivicGateway*	Encrypted PII data, UUID
ValidationResult	*CivicValidator* finishes validating a ValidationRequest.	Decision data, UUID
DeletionRequest	User sends POST request to *CivicGateway*	UUID
DataStored	*CivicPIIStorage* has stored PII in the HighSecDataStore.	UUID

Services

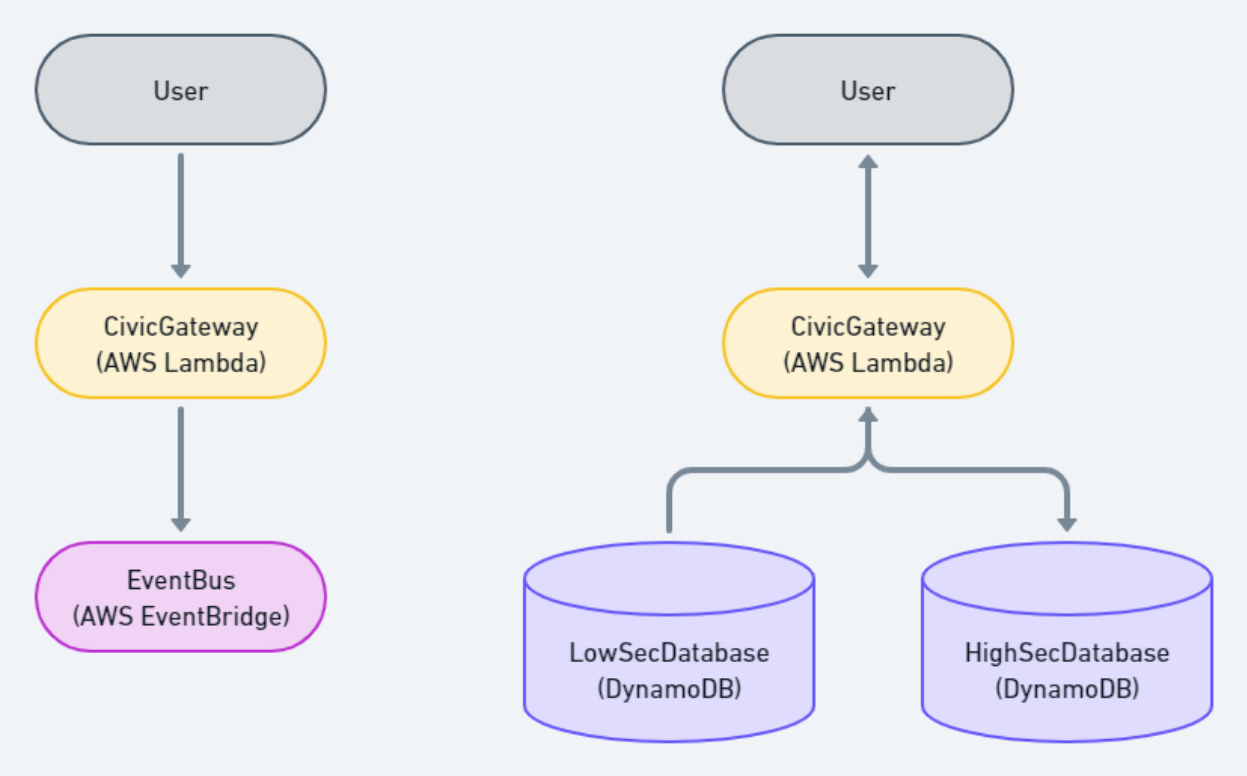
CivicGateway - AWS Lambda

CivicGateway is responsible for handling any incoming and outgoing requests. It exposes 3 endpoints to the user:

Method	Description
GET	Used for fetching the current status of a validation request (payload is UUID).
POST	Used for creating a new validation request (payload is PII data). - Generates a UUIDs. - Encrypts/decrypts PII using AWS KMS. - Publishes ValidationReques.
DELETE	Used for deleting decision/PII data.

CivicGateway is responsible encrypting/decrypting PII by using AWS KMS. In the SAM provisioning template we expose CivicKey (CMK) to this service which can be used for encrypting/decrypting data.

CivicGateway is also responsible for generating UUID's which it associates with encrypted PII data before it passes the data on to other services.

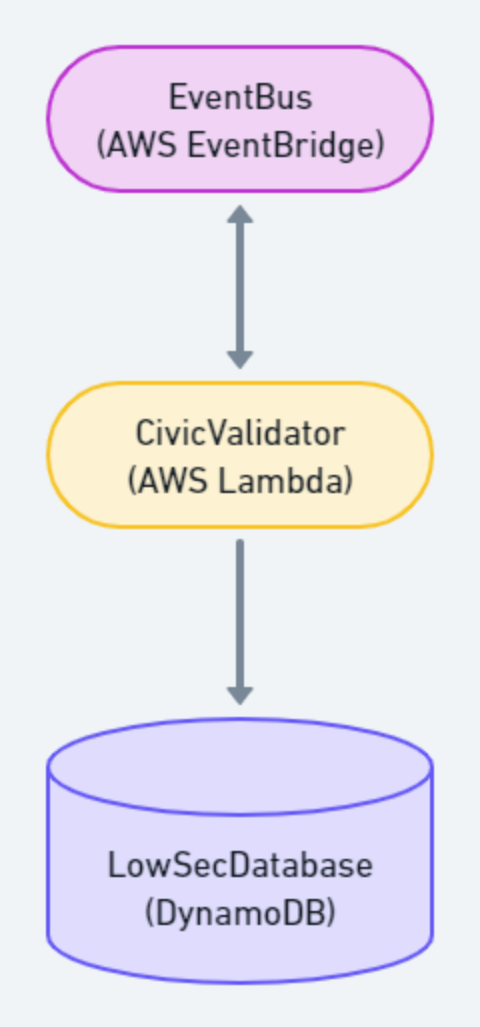


CivicValidator - AWS Lambda

CivicValidator is responsible for decrypting ValidationRequests, then validation those and lastly storing decision data into the LowSecDataStore. It subscribes to the **ValidationRequest** and **DeletionRequest** events:

Event	Description
ValidationRequest	Read UUID and PII from event payload, decrypt PII, perform validation on it and insert into LowSecDataStore.
DeletionRequest	Read UUID from event payload and use it to delete a record from LowSecDataStore.

After validation is complete CivicValidator will publish a **ValidationResult** of which the payload will contain UUID and decision data.

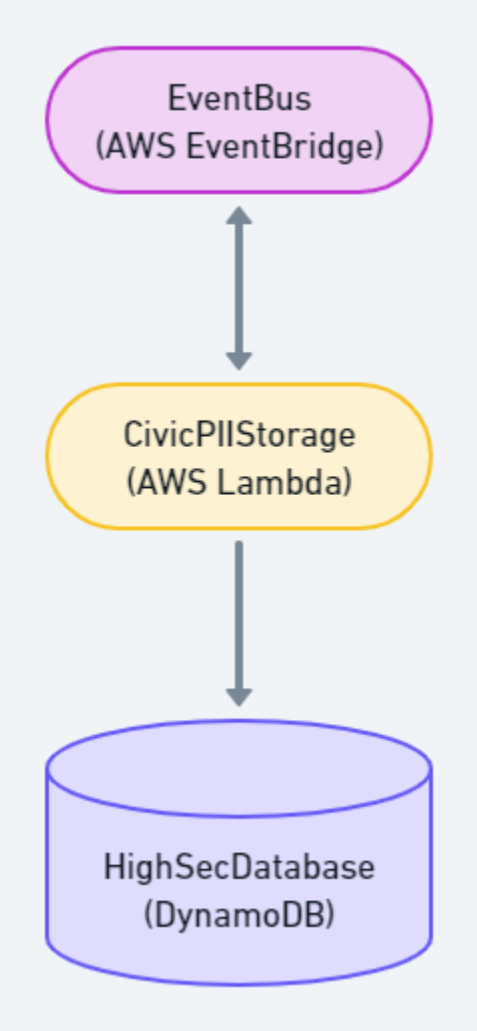


CivicPIIStorage - AWS Lambda

CivicPIIStorage is responsible for storing/updating/deleting PII. It subscribes to **ValidationRequest** and **DeletionRequest** events:

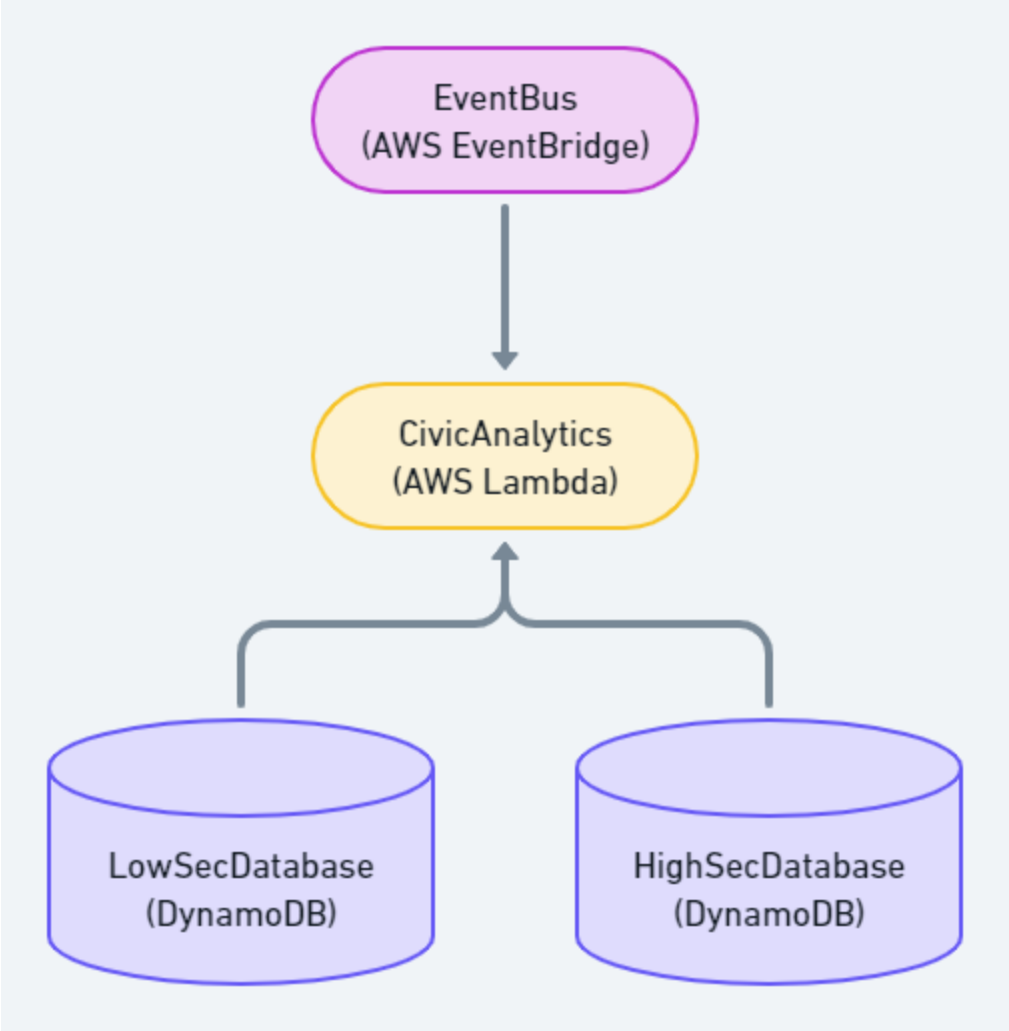
Event	Description
ValidationRequest	Read UUID and PII data from event payload, store PII data into HighSecDataStore using UUID as key.
DeletionRequest	Read UUID from event payload, remove record from HighSecDataStore using UUID as key.

After storing, updating deleting of PII is finished CivicPIIStorage will publish a DataStored, DataUpdated or DataDeleted event respectively. Payload of these events is the UUID.



CivicAnalytics - AWS Lambda

CivicAnalytics is responsible for aggregating all necessary data and continuing with the analytics pipeline. It subscribes to all events that are being published onto the event bus and based on a set of rules determines what data to aggregate and push forward into the pipeline. Data aggregation is possible through allowing reads from Low- and HighSecDataStores.



DataStores

HighSecDataStore - DynamoDB

This datastore is using DynamoDB, it is used for storing sensitive PII. The data is stored in encrypted state, by using this approach a malicious actor who gains access to the database is not able to retrieve sensitive data.

Name	Type	Description	Primary key
id	String	UUID	Yes
username	String	Encrypted username	No
full_name	String	Encrypted full_name	No
date_of_birth	String	Encrypted date_of_birth	No

Name	Type	Description	Primary key
address	String	Encrypted address	No
phone_number	String	Encrypted phone_number	No
created	Date	ISO_8601 Date string for when record was created	No
updated	Date	ISO_8601 Date string for when record was last updated	No

LowSecDataStore - DynamoDB

Name	Type	Description	Primary key
id			
decision	Number	Boolean that indicates whether validation was accepted or rejected.	No
status	Number	Boolean that indicates the current status of the validation request (pending or fulfilled).	No
created	Date	ISO_8601 Date string for when record was created	No
updated	Date	ISO_8601 Date string for when record was last updated	No

Access Control

Service	Database permissions	Other permissions
CivicGateway	DBGet LowSecDataStore DBGet HighSecDataStore	Publish onto EventBridge Encrypt/decrypt using CivicKey (CMK)
CivicValidator	DBPut LowSecDataStore DBUpdate LowSecDataStore DBDelete LowSecDataStore	Publish/subscribe EventBridge Decrypt using CivicKey (CMK)
CivicPIIStorage	DBPut HighSecDataStore DBUpdate HighSecDataStore DBDelete HighSecDataStore	Publish/subscribe EventBridge
CivicAnalytics	DBGet LowSecDataStore DBGet HighSecDataStore	Decrypt using CivicKey (CMK)

Security

In order for our system to be as secure as possible I propose we encrypt the data as soon as it arrives. By using *AWS KMS* we are able to encrypt the individual fields within a payload when it enters our system thereby minimizing the risk when one of the services is compromised but preserving the PII data structure. Services that rely on sensitive data (such as *CivicValidator*) will be able to access said data in **isolation** through direct communication with *AWS KMS* for decryption. Using this approach we minimize moments where sensitive data is in a decrypted state. I go into more detail in the System Overview section.

In addition to having the sensitive data being in encrypted state for most of the in-flight moments we also store the data in encrypted state for maximum security. We achieve this by generating a UUID when a ValidationRequest enters our system. This UUID is then associated with the encrypted data throughout the validation process and across services.

GDPR

Security/safety

In order to comply with GDPR's security/safety clause we store our data in encrypted state in our *HighSecDataStore*. By using this approach we make sure that no sensitive data is lost if a malicious actor gains access to our database. Furthermore, as stated before, the PII will be in encrypted state for almost all of the in-flight moments.

Access/review

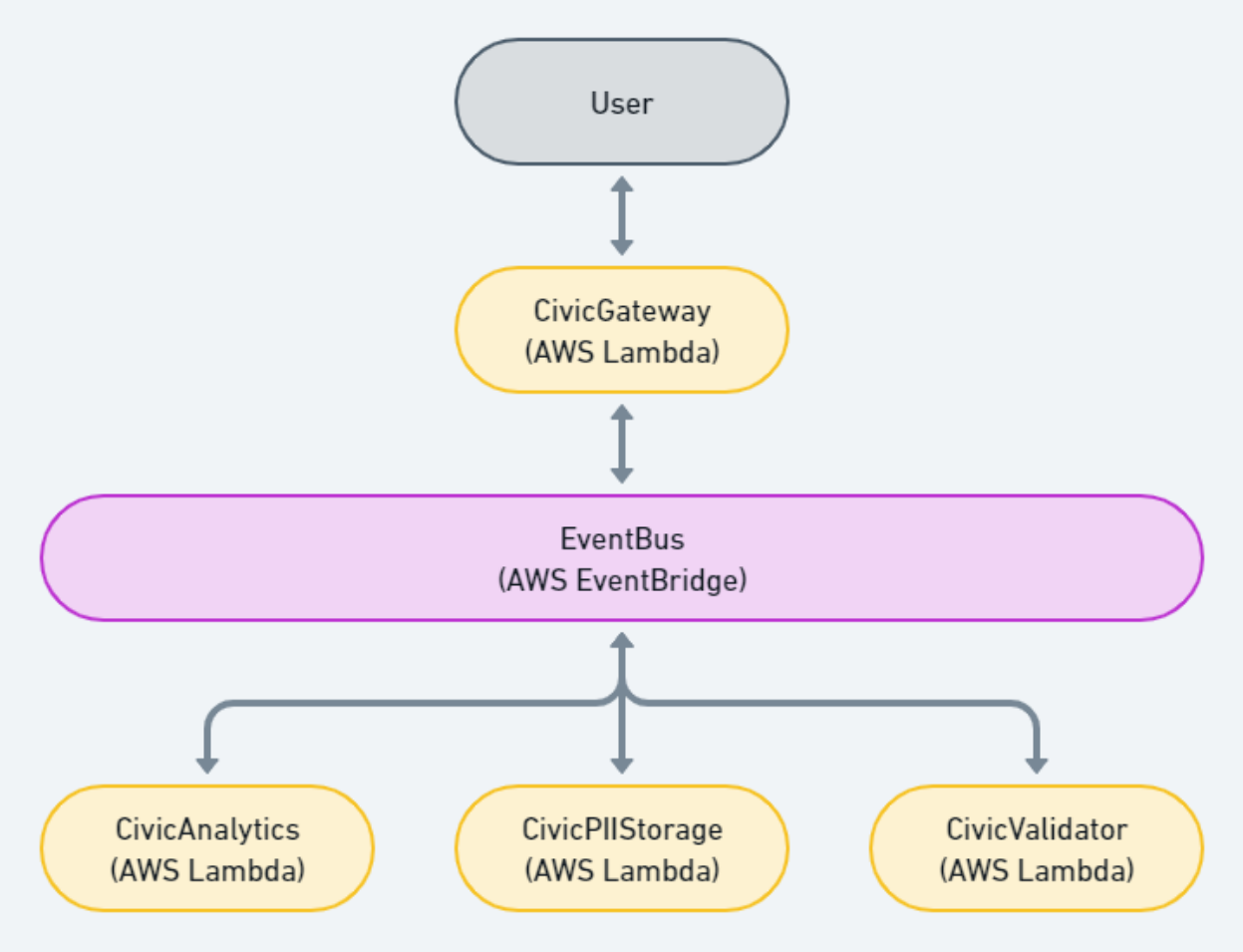
GDPR states that users should be able to access/review their personal data. To accomodate for this *CivicGateway* has a /GET method that will retrieve the PII data from the *HighSecDataStore*, decrypt it and send it as payload in a response.

Deletion

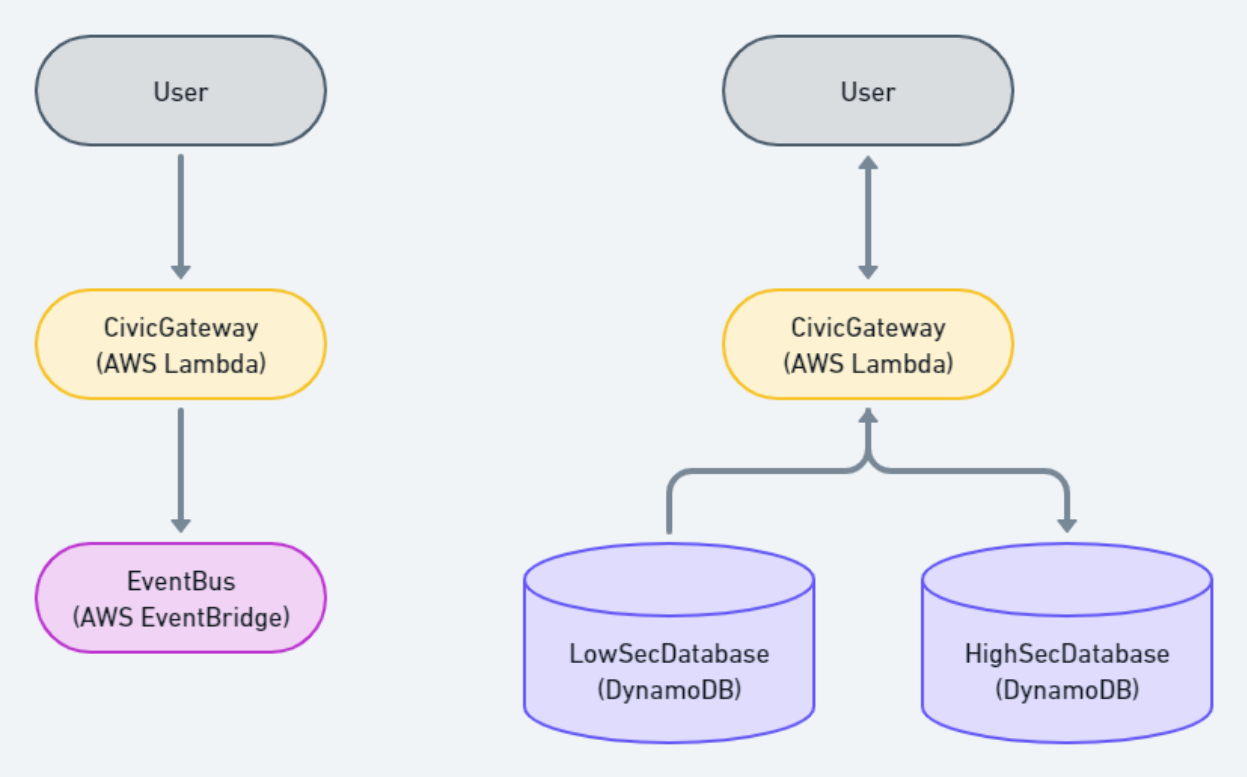
In order to comply with the users' right to request data to be forgotten I have exposed a /DELETE endpoint. This endpoint will remove the decision and PII from our databases. However, one could argue that the decision data could be kept since it does not contain any sensitive data.

Diagrams

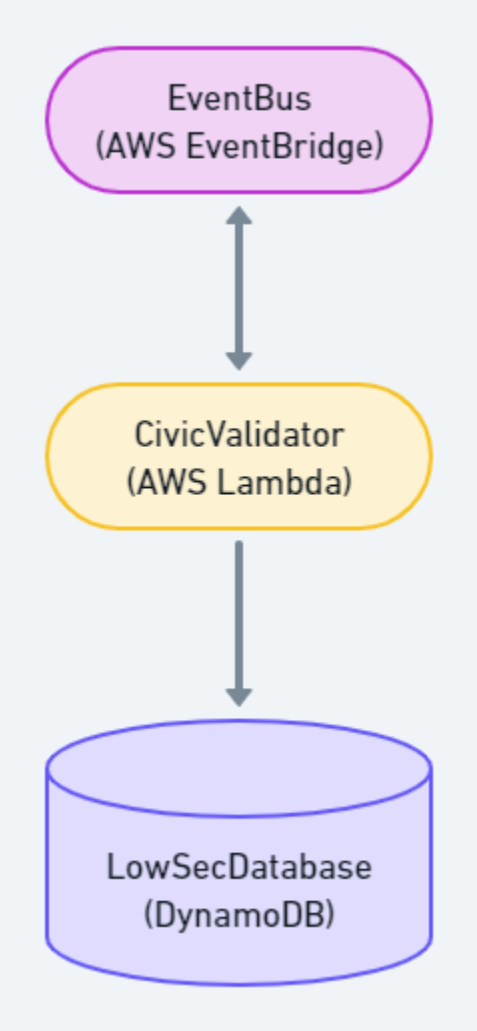
High level overview



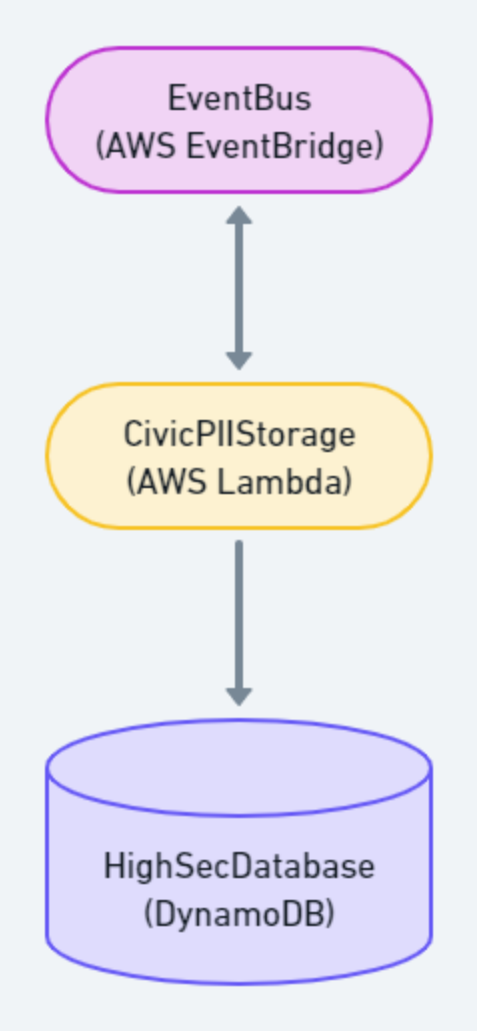
CivicGateway



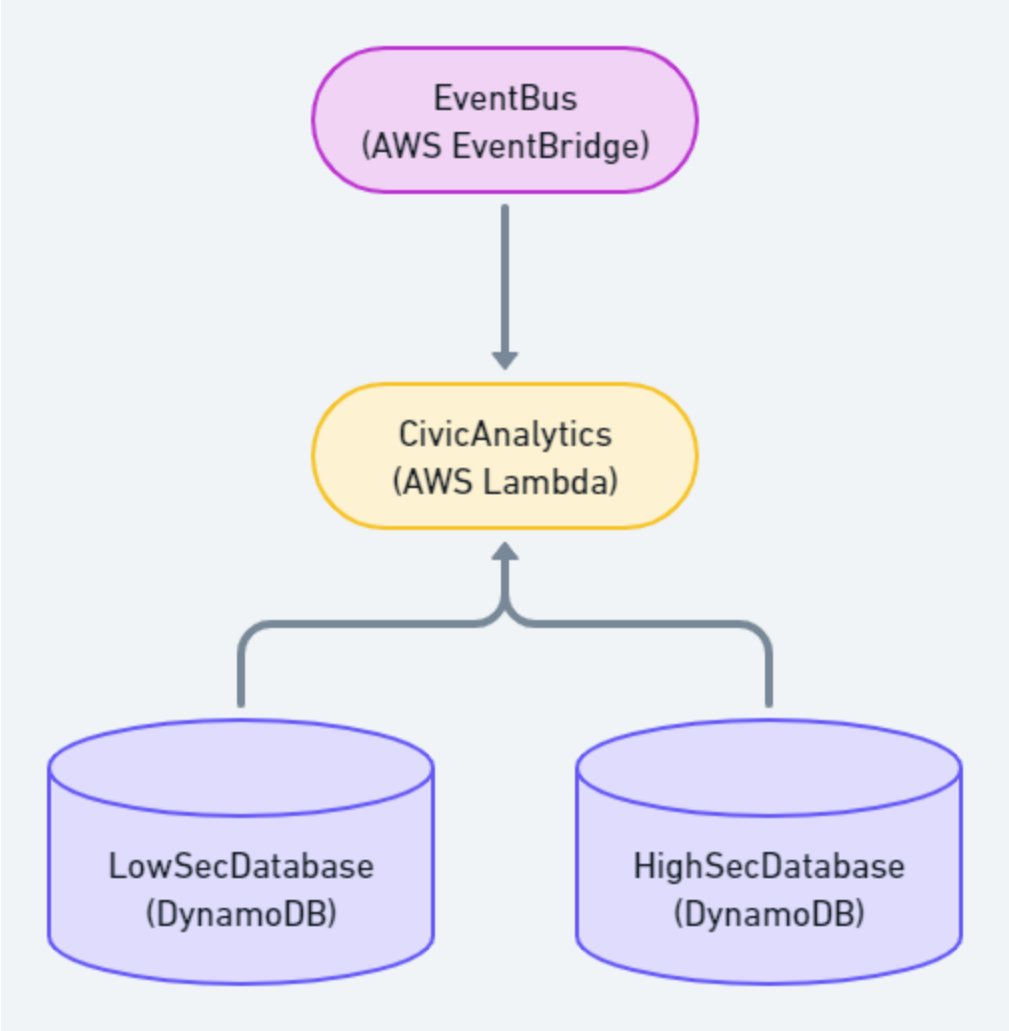
CivicValidator



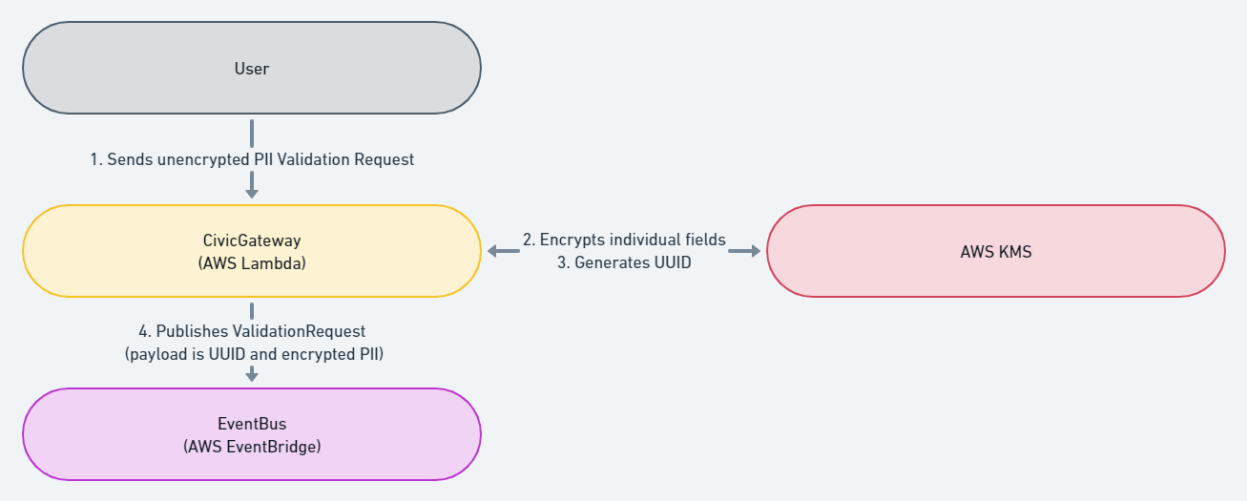
CivicPIIStorage



CivicAnalytics

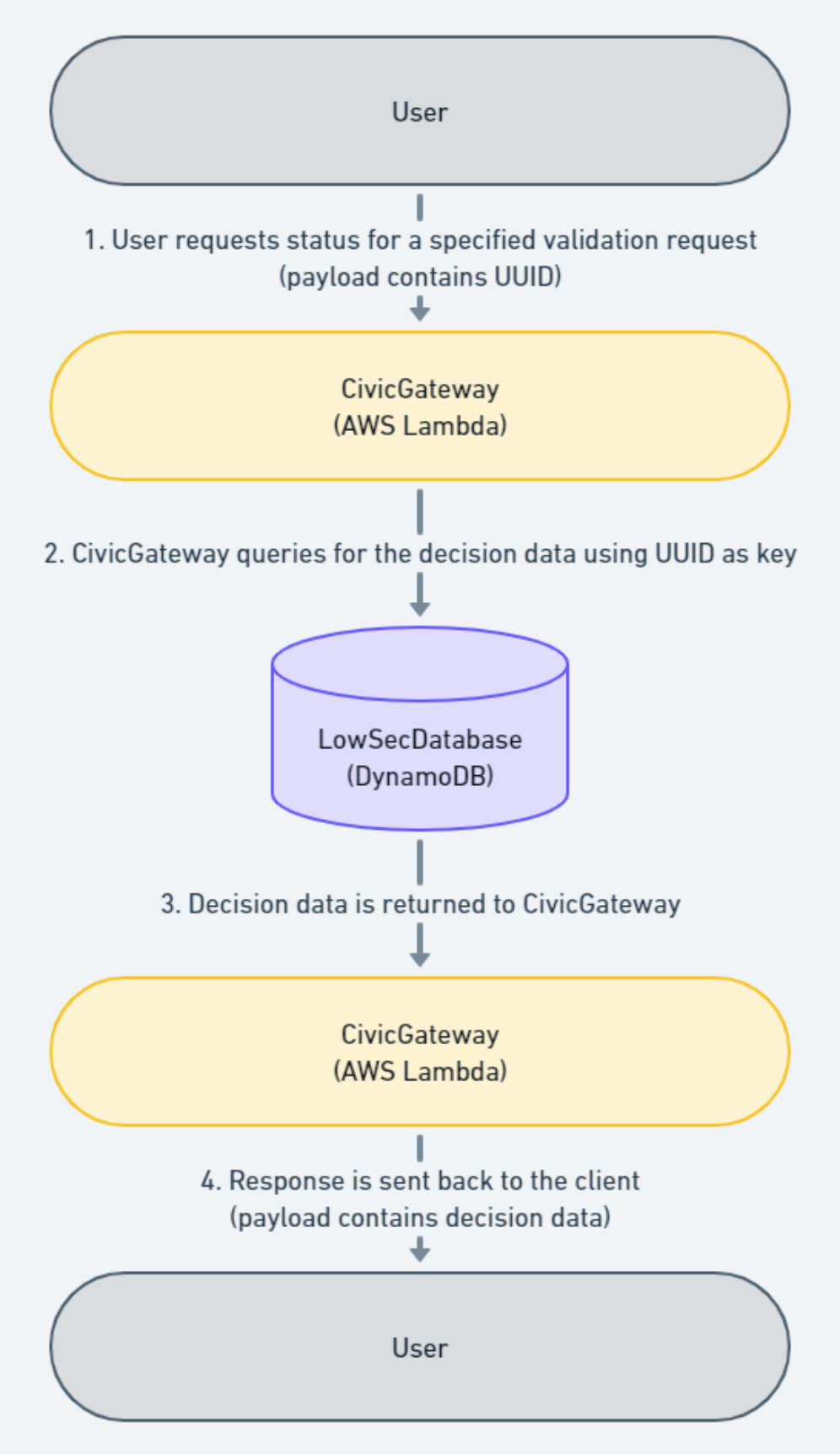


ValidationsRequests



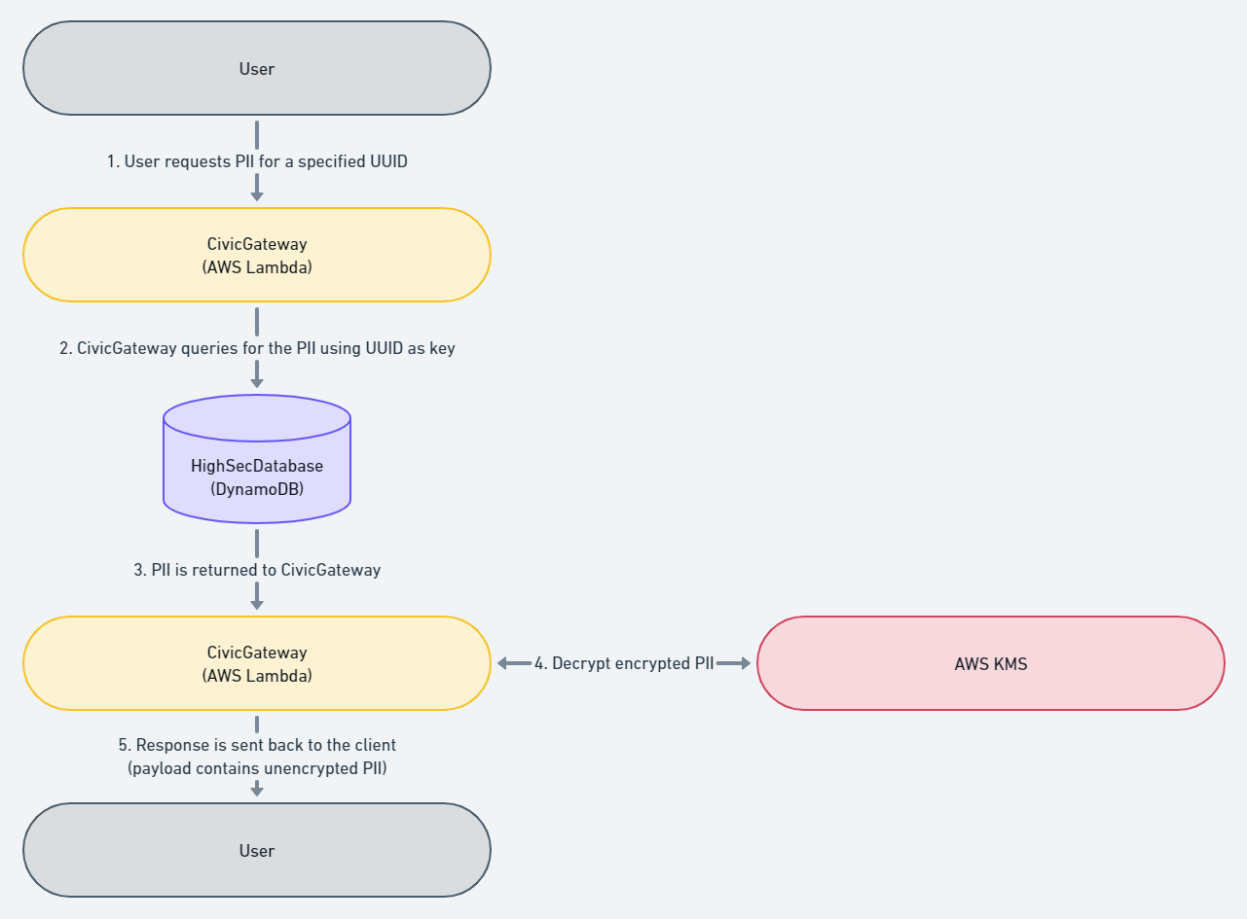
1. User sends PII Validation Request to *CivicGateway*.
2. *CivicGateway* encrypts individual PII fields using *AWS KMS*.
3. *CivicGateway* also generates a UUID and associates it with the encrypted PII.
4. *CivicGateway* emits a **ValidationRequest** through *EventBridge* where the payload consists of UUID and encrypted PII.

Validation status/decision requests


User
1. User requests status for a specified validation request
(payload contains UUID)
CivicGateway
(AWS Lambda)
2. CivicGateway queries for the decision data using UUID as key
LowSecDatabase
(DynamoDB)
3. Decision data is returned to CivicGateway
CivicGateway
(AWS Lambda)
4. Response is sent back to the client
(payload contains decision data)
User

1. User requests status for a specified validation request by providing UUID.
2. CivicGateway queries DecisionDataStore using UUID as key (Caching also an option).
3. Status / decision data is returned to CivicGateway.
4. Response is sent back to the client where payload contains status / decision data.

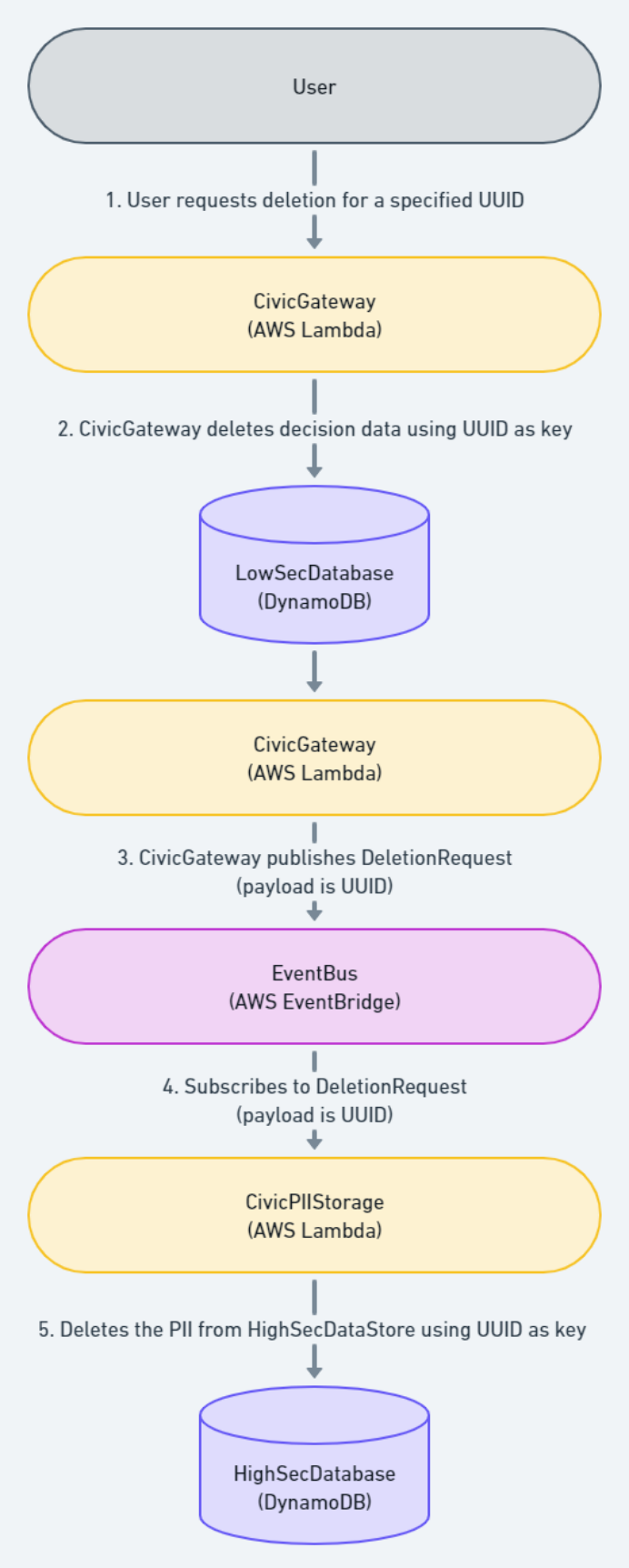
PII Requests



1. User requests PII for a specified UUID.
2. CivicGateway queries the PII data from HighSecDataStore using the UUID as key.
3. PII is returned in encrypted state to CivicGateway.
4. CivicGateway uses AWS KMS to decrypt the PII.

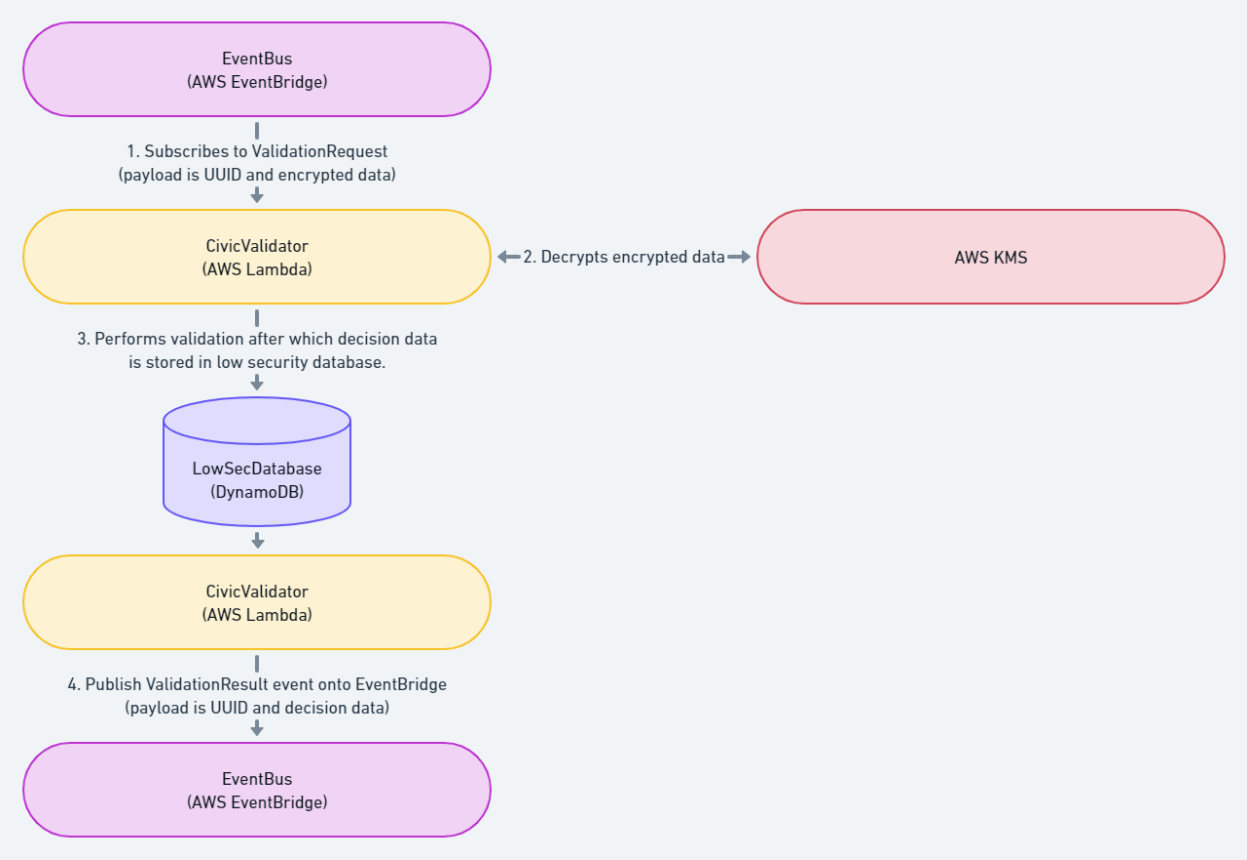
5. PII is sent to the user via the response.

DeletionRequests


User
1. User requests deletion for a specified UUID
CivicGateway
(AWS Lambda)
2. CivicGateway deletes decision data using UUID as key
LowSecDatabase
(DynamoDB)
CivicGateway
(AWS Lambda)
3. CivicGateway publishes DeletionRequest
(payload is UUID)
EventBus
(AWS EventBridge)
4. Subscribes to DeletionRequest
(payload is UUID)
CivicPIIStorage
(AWS Lambda)
5. Deletes the PII from HighSecDataStore using UUID as key
HighSecDatabase
(DynamoDB)

1. User requests deletion for a specified UUID.
2. CivicGateway deletes decision data from LowSecDataStore using UUID as key.
3. CivicGateway pulishes DeletionRequest where payload contains UUID.
4. CivicPIIStorage subscribes to DeletionRequest (payload contains UUID).
5. CivicPIIStorage deletes the PII from HighSecDataStore using UUID as key.

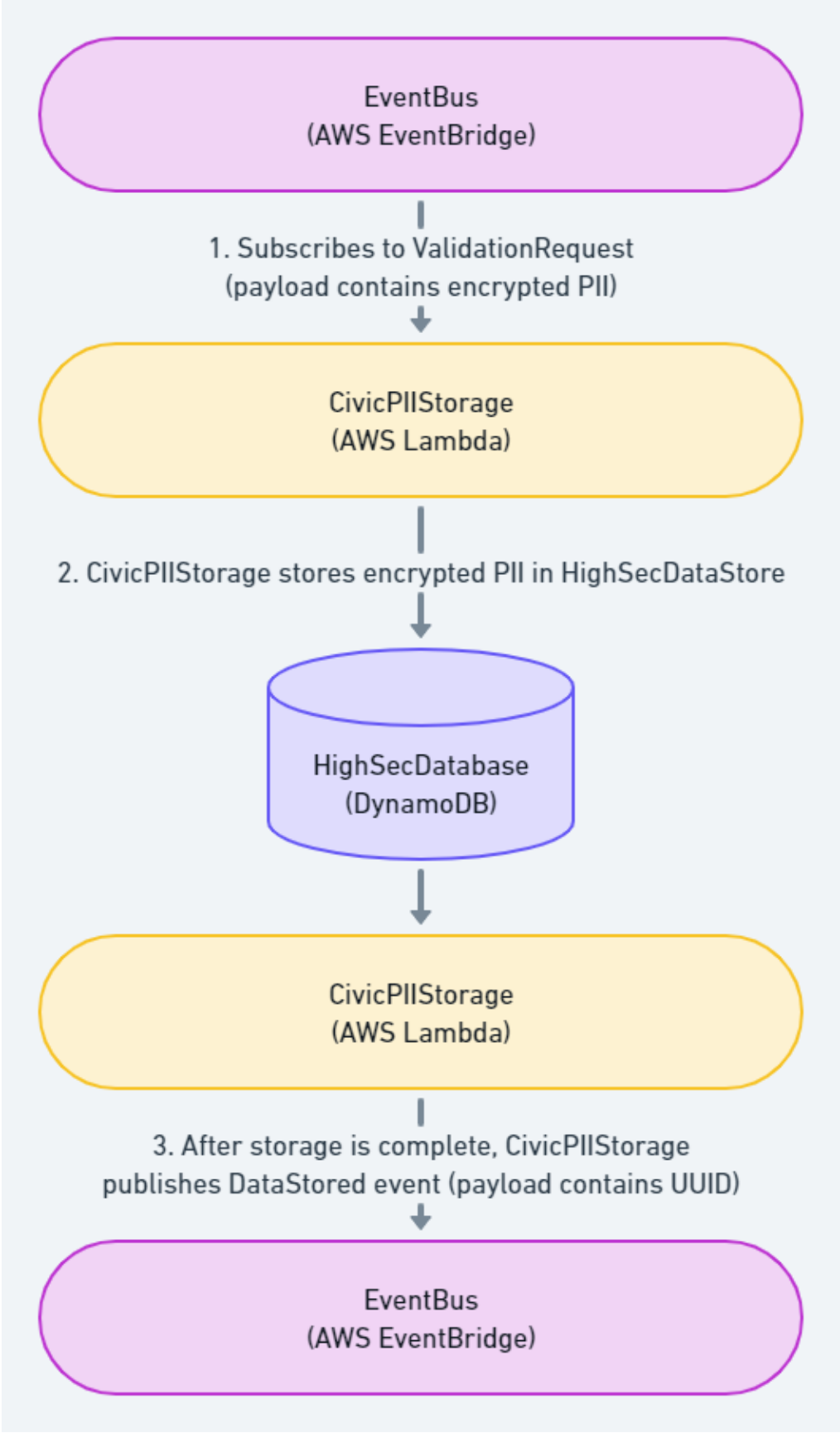
Validation



1. *CivicValidator* subscribes to *EventBridge* and receives **ValidationRequest** event. **ValidationRequest** contains UUID and encrypted PII.
2. *CivicValidator* uses *AWS KMS* to decrypt the encrypted PII.
3. *CivicValidator* performs validation on unencrypted PII and stores decision data in low security datastore.

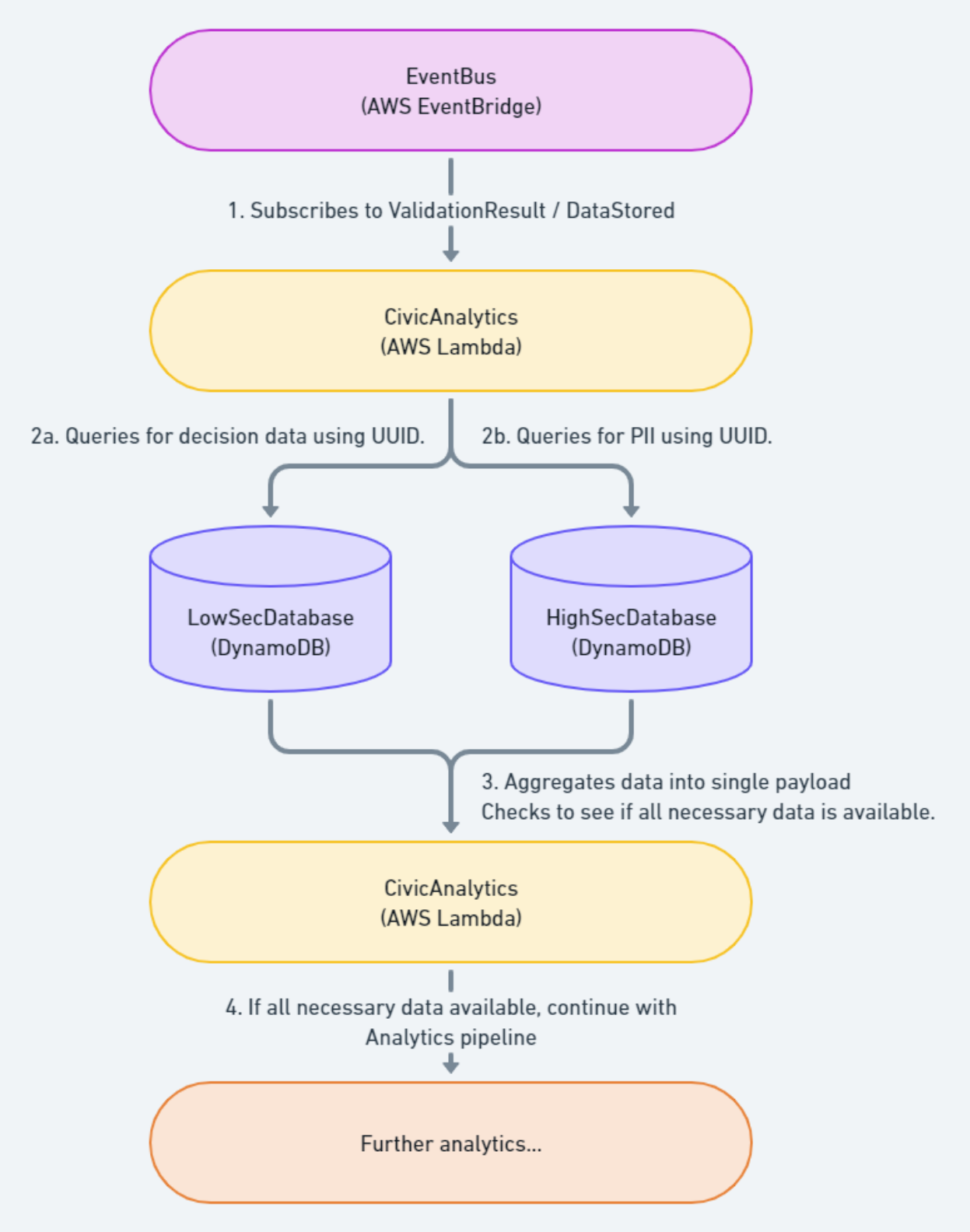
4. *CivicValidator* publishes **ValidationResult** event to *EventBridge* when validation finishes. UUID is the payload along with decision data.

PII Storage


EventBus
(AWS EventBridge)
1. Subscribes to ValidationRequest
(payload contains encrypted PII)
CivicPIIStorage
(AWS Lambda)
2. CivicPIIStorage stores encrypted PII in HighSecDataStore
HighSecDatabase
(DynamoDB)
CivicPIIStorage
(AWS Lambda)
3. After storage is complete, CivicPIIStorage
publishes DataStored event (payload contains UUID)
EventBus
(AWS EventBridge)

1. *CivicPIIStorage* subscribes to **ValidationRequest** event.
   ValidationRequest event contains UUID and encrypted PII.
2. *CivicPIIStorage* stores encrypted PII in high security data store.
3. *CivicPIIStorage* publishes **DataStored** event.
   DataStored event has UUID as payload.

Aggregating data for analytics pipeline


EventBus
(AWS EventBridge)
1. Subscribes to ValidationResult / DataStored
CivicAnalytics
(AWS Lambda)
2a. Queries for decision data using UUID.
2b. Queries for PII using UUID.
LowSecDatabase
(DynamoDB)
HighSecDatabase
(DynamoDB)
3. Aggregates data into single payload
Checks to see if all necessary data is available.
CivicAnalytics
(AWS Lambda)
4. If all necessary data available, continue with
Analytics pipeline
Further analytics...

1. *CivicAnalytics* subscribes to **ValidationResult** and **DataStored** events.
   Payloads contains UUID.
2. *CivicAnalytics* uses UUID to retrieve decision and PII data from low & high security datastores.
3. *CivicAnalytics* aggregates data into a single payload and performs check to see if all necessary data is available.
4. *CivicAnalytics* continues with analytics pipeline if all necessary data is available.