PrefMan

A user preference management solution

The Problem

A solution that centralizes user preferences

Focus areas for role & case

- From case:
  - Security, scalability, AWS, IaC (CDK)
- From interviews:
  - AWS, serverless, AWS, IaC (CloudFormation/CDK)

Approach

- Saw this as a learning opportunity
- Wanted to make a fully functional MVP
  - Focus on core features, along with the focus areas of the case
- Started off by refreshing my AWS knowledge
  - Looked into possible solutions for serverless computing and databases, and what would make sense for the case
- Familiared myself with CDK
  - Set up a few example stacks to test out
- Considered data model and DB options

Data model

What should the external systems receive?


```
{
    "userId": "8c9b209e-6275-4b85-9cda-ddf6a3964cd8",
    "email": "user@email.com",
    "cookiePreferences": {
        "analytics": true,
        "internalMarketing": true,
        "thirdPartyMarketing": false
    },
    "interfacePreferences": {
        "darkMode": true,
        "preferDesktopOnMobile": true
    },
    "managedAccounts": [
        {
            "userId": "d51ff7b6-6063-4474-a415-02a42f56ed05",
            "participationConsentGiven": true
        },
        {
            "userId": "fca272cd-fc94-4435-8a16-2ce1c398535b",
            "participationConsentGiven": false
        }
    ]
}
```



```
{
    "UserId": "github25583215",
    "ManagedByUserId": null,
    "OwnPreferences": [
        {
            "PreferenceValue": "example@github.com",
            "FriendlyName": "Preferred e-mail",
            "LogicalName": "Contact.Email",
            "Type": "string",
            "PreferenceId": "7ce10d11-733f-4ade-ba06-483ceb235374",
            "UpdatedAt": "2023-02-07 11:53:21 +00:00"
        },
        {
            "PreferenceValue": false,
            "FriendlyName": "Analytics cookies",
            "LogicalName": "Cookies.Analytics",
            "Type": "bool",
            "PreferenceId": "462f44ce-e97a-4e42-946e-eb37b11d58fa",
            "UpdatedAt": "2023-02-07 11:53:21 +00:00"
        },
        {
            "PreferenceValue": false,
            "FriendlyName": "Internal marketing cookies",
            "LogicalName": "Cookies.InternalMarketing",
            "Type": "bool",
            "PreferenceId": "89563418-53d2-4bc3-b71a-5419341a3386",
            "UpdatedAt": "2023-02-07 11:53:21 +00:00"
        },
        {
            "PreferenceValue": false,
            "FriendlyName": "Third-party marketing cookies",
            "LogicalName": "Cookies.ThirdPartyMarketing",
            "Type": "bool",
            "PreferenceId": "e368c2b0-5e33-4207-9141-a3b516369b92",
            "UpdatedAt": "2023-02-07 11:53:21 +00:00"
        },
        {
            "PreferenceValue": true,
            "FriendlyName": "Dark mode interface",
            "LogicalName": "InterfacePreferences.DarkMode",
            "Type": "bool",
            "PreferenceId": "17a40602-71a8-41ff-b12c-af6e9b3bd69c",
            "UpdatedAt": "2023-02-07 11:53:21 +00:00"
        },
        {
            "PreferenceValue": false,
            "FriendlyName": "Use desktop site on mobile",
            "LogicalName": "InterfacePreferences.PreferDesktopOnMobile",
            "Type": "bool",
            "PreferenceId": "4abfaa90-c540-4e3e-9dce-911beebef22c",
            "UpdatedAt": "2023-02-07 11:53:21 +00:00"
        }
    ],
    "ManagingPreferences": [
        {
            "ManagedForUserId": "google-oauth2101466600451189785080",
            "PreferenceValue": true,
            "FriendlyName": "Consent of participation in competitions",
            "LogicalName": "ManagingPreferences.ParticipationConsentGiven",
            "Type": "bool",
            "PreferenceId": "b75fcff3-f026-4d0d-b390-b26123eae56d",
            "UpdatedAt": "2023-02-07 11:53:21 +00:00"
        }
    ],
    "ManagedPreferences": []
}
```


- Challenges:
  - Implications of admin actions
  - Saving ”list of objects” and handling relations
    - SQL vs Document vs Key-Value

Admin functionality

- What does "manage" mean in this case?
- My assumption was that admins can edit preference metadata:
  - Friendly name, logical name, type, default value, and whether it is a "managed" preference (i.e. between parent and child user)
- Also add new possible preferences, and disable or delete current ones.
- Currently not managing user preference values, but that is easy to add (update authorization logic for user preference APIs)

So, what about the database?

- SQL vs NoSQL
  - NoSQL scalable horizontally, multi-region
  - There is some relational data with "managed" preferences (parent-child)
  - Wanted more experience with NoSQL
- NoSQL: Document vs Key-Value
  - DocumentDB vs **DynamoDB**
  - Serverless, scalability
  - Model/schema not that complex… or is it?

Preference metadata

```
[DynamoDBTable("PreferenceMetadata")]
49 references
public class PreferenceMetadata
{
    [DynamoDBHashKey]
    24 references
    public string PreferenceId { get; set; }

    12 references
    public string LogicalName { get; set; }

    11 references
    public string FriendlyName { get; set; }

    9 references
    public string Type { get; set; }

    13 references
    public string DefaultValue { get; set; }

    13 references
    public bool Enabled { get; set; }

    10 references
    public bool IsManaged { get; set; }
}
```

PreferenceId	DefaultValue	Enabled	FriendlyName	IsManaged	LogicalName	Type
7ce10d11-733f-4ade-...		1	Contact e-mail	0	Contact.Email	string
b75fcff3-f026-4d0d-...		1	Consent of par...	1	ManagingPref...	bool
17a40602-71a8-41ff-...	true	1	Dark mode inte...	0	InterfacePref...	bool
4abfaa90-c540-4e3e-...	false	1	Use desktop sit...	0	InterfacePref...	bool

```
[DynamoDBTable("UserPreferences")]
8 references
public class UserPreferencesDynamo
{
    [DynamoDBHashKey]
    2 references
    public string UserId { get; set; }

    2 references
    public string ManagedByUserId { get; set; }

    2 references
    public List<Dictionary<string, string>> OwnPreferences { get; set; } = new

    3 references
    public List<Dictionary<string, string>> ManagingPreferences { get; set; }
}
```

DB model

```
28 references
public class UserPreferences
{
    9 references
    public string UserId { get; set; }

    8 references
    public string ManagedByUserId { get; set; }

    20 references
    public List<BasePreference> OwnPreferences { get; set;

    13 references
    public List<ManagingPreference> ManagingPreferences { g
}

21 references
public class BasePreference
{
    24 references
    public string PreferenceId { get; set; }

    18 references
    public string PreferenceValue { get; set; }

    14 references
    public string UpdatedAt { get; set; }
}

12 references
public class ManagingPreference : BasePreference
{
    5 references
    public string ManagingForUserId { get; set; }
}
```

Logical model
PUT API contract

Enriched response for GET /UserPreferences
(added metadata and relational data)

```
5 references
public class EnrichedUserPreferences
{
    1 reference
    public string UserId { get; set; }

    1 reference
    public string ManagedByUserId { get; set; }

    2 references
    public List<EnrichedBasePreference> OwnPreferences { get; set; } =

    2 references
    public List<EnrichedManagingPreference> ManagingPreferences { get;

    1 reference
    public List<EnrichedBasePreference> ManagedPreferences { get; set;
}

9 references
public class EnrichedBasePreference : BasePreference
{
    5 references
    public new dynamic PreferenceValue { get; set; }

    6 references
    public string FriendlyName { get; set; }

    2 references
    public string LogicalName { get; set; }

    2 references
    public string Type { get; set; }
}

4 references
public class EnrichedManagingPreference : EnrichedBasePreference
{
    2 references
    public string ManagedForUserId { get; set; }
}
```

```
5 references
public class EnrichedUserPreferences
{
    1 reference
    public string UserId { get; set; }

    1 reference
    public string ManagedByUserId { get; set; }

    2 references
    public List<EnrichedBasePreference> OwnPreferences { get; set; } =

    2 references
    public List<EnrichedManagingPreference> ManagingPreferences { get;

    1 reference
    public List<EnrichedBasePreference> ManagedPreferences { get; set;
}

9 references
public class EnrichedBasePreference : BasePreference
{
    5 references
    public new dynamic PreferenceValue { get; set; }

    6 references
    public string FriendlyName { get; set; }

    2 references
    public string LogicalName { get; set; }

    2 references
    public string Type { get; set; }
}

4 references
public class EnrichedManagingPreference : EnrichedBasePreference
{
    2 references
    public string ManagedForUserId { get; set; }
}
```


```
{
    "UserId": "github25583215",
    "ManagedByUserId": null,
    "OwnPreferences": [
        {
            "PreferenceValue": "example@github.com",
            "FriendlyName": "Preferred e-mail",
            "LogicalName": "Contact.Email",
            "Type": "string",
            "PreferenceId": "7ce10d11-733f-4ade-ba06-483ceb235374",
            "UpdatedAt": "2023-02-07 11:53:21 +00:00"
        },
        {
            "PreferenceValue": false,
            "FriendlyName": "Analytics cookies",
            "LogicalName": "Cookies.Analytics",
            "Type": "bool",
            "PreferenceId": "462f44ce-e97a-4e42-946e-eb37b11d58fa",
            "UpdatedAt": "2023-02-07 11:53:21 +00:00"
        },
        {
            "PreferenceValue": false,
            "FriendlyName": "Internal marketing cookies",
            "LogicalName": "Cookies.InternalMarketing",
            "Type": "bool",
            "PreferenceId": "89563418-53d2-4bc3-b71a-5419341a3386",
            "UpdatedAt": "2023-02-07 11:53:21 +00:00"
        },
        {
            "PreferenceValue": false,
            "FriendlyName": "Third-party marketing cookies",
            "LogicalName": "Cookies.ThirdPartyMarketing",
            "Type": "bool",
            "PreferenceId": "e368c2b0-5e33-4207-9141-a3b516369b92",
            "UpdatedAt": "2023-02-07 11:53:21 +00:00"
        },
        {
            "PreferenceValue": true,
            "FriendlyName": "Dark mode interface",
            "LogicalName": "InterfacePreferences.DarkMode",
            "Type": "bool",
            "PreferenceId": "17a40602-71a8-41ff-b12c-af6e9b3bd69c",
            "UpdatedAt": "2023-02-07 11:53:21 +00:00"
        },
        {
            "PreferenceValue": false,
            "FriendlyName": "Use desktop site on mobile",
            "LogicalName": "InterfacePreferences.PreferDesktopOnMobile",
            "Type": "bool",
            "PreferenceId": "4abfaa90-c540-4e3e-9dce-911beebef22c",
            "UpdatedAt": "2023-02-07 11:53:21 +00:00"
        }
    ],
    "ManagingPreferences": [
        {
            "ManagedForUserId": "google-oauth2|101466600451189785080",
            "PreferenceValue": true,
            "FriendlyName": "Consent of participation in competitions",
            "LogicalName": "ManagingPreferences.ParticipationConsentGiven",
            "Type": "bool",
            "PreferenceId": "b75fcff3-f026-4d0d-b390-b26123eae56d",
            "UpdatedAt": "2023-02-07 11:53:21 +00:00"
        }
    ],
    "ManagedPreferences": []
}
```

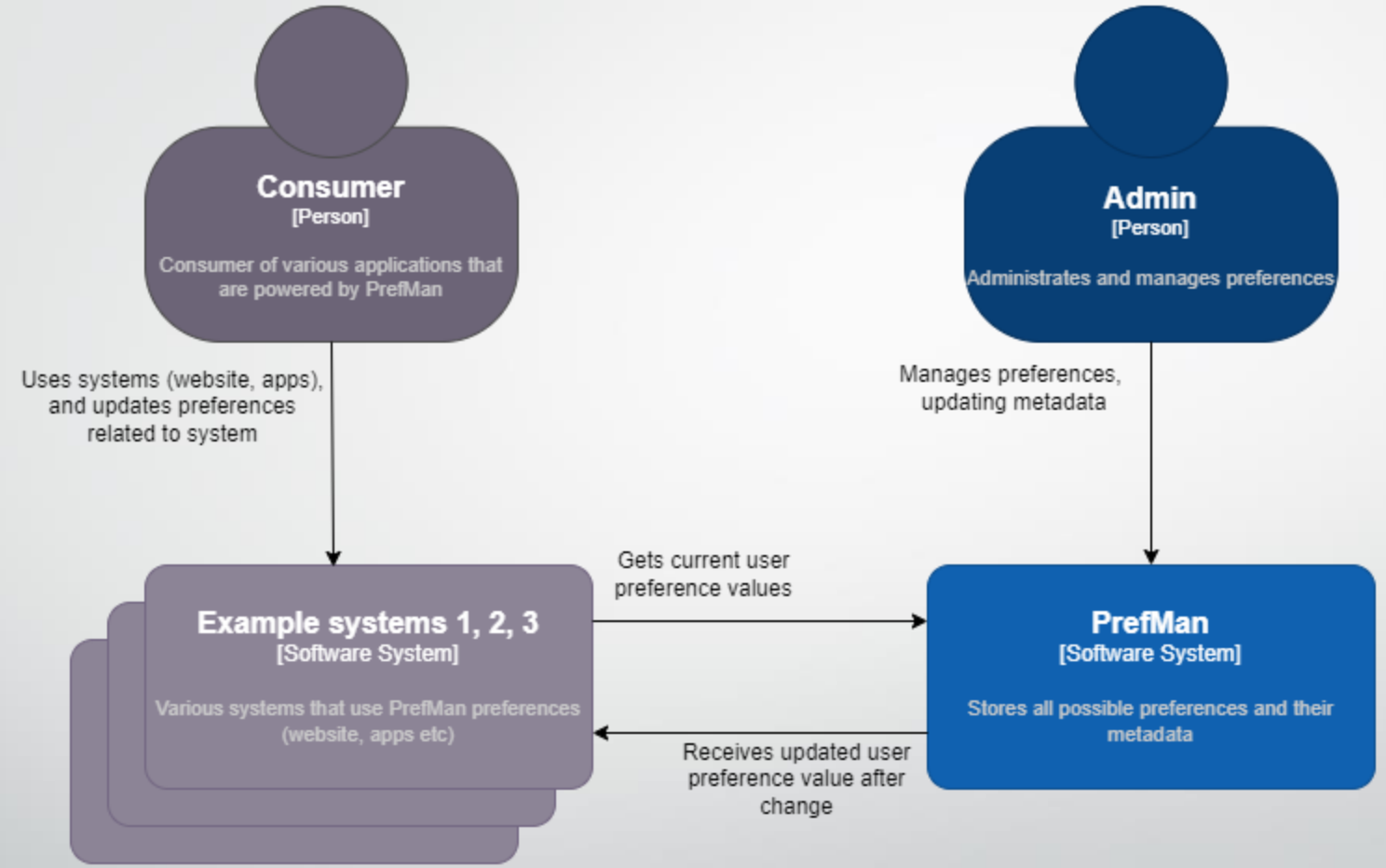
Infrastructure

- EC2, Fargate or Lambda?
- **Serverless, multi-region, scalability**
- Learning opportunity

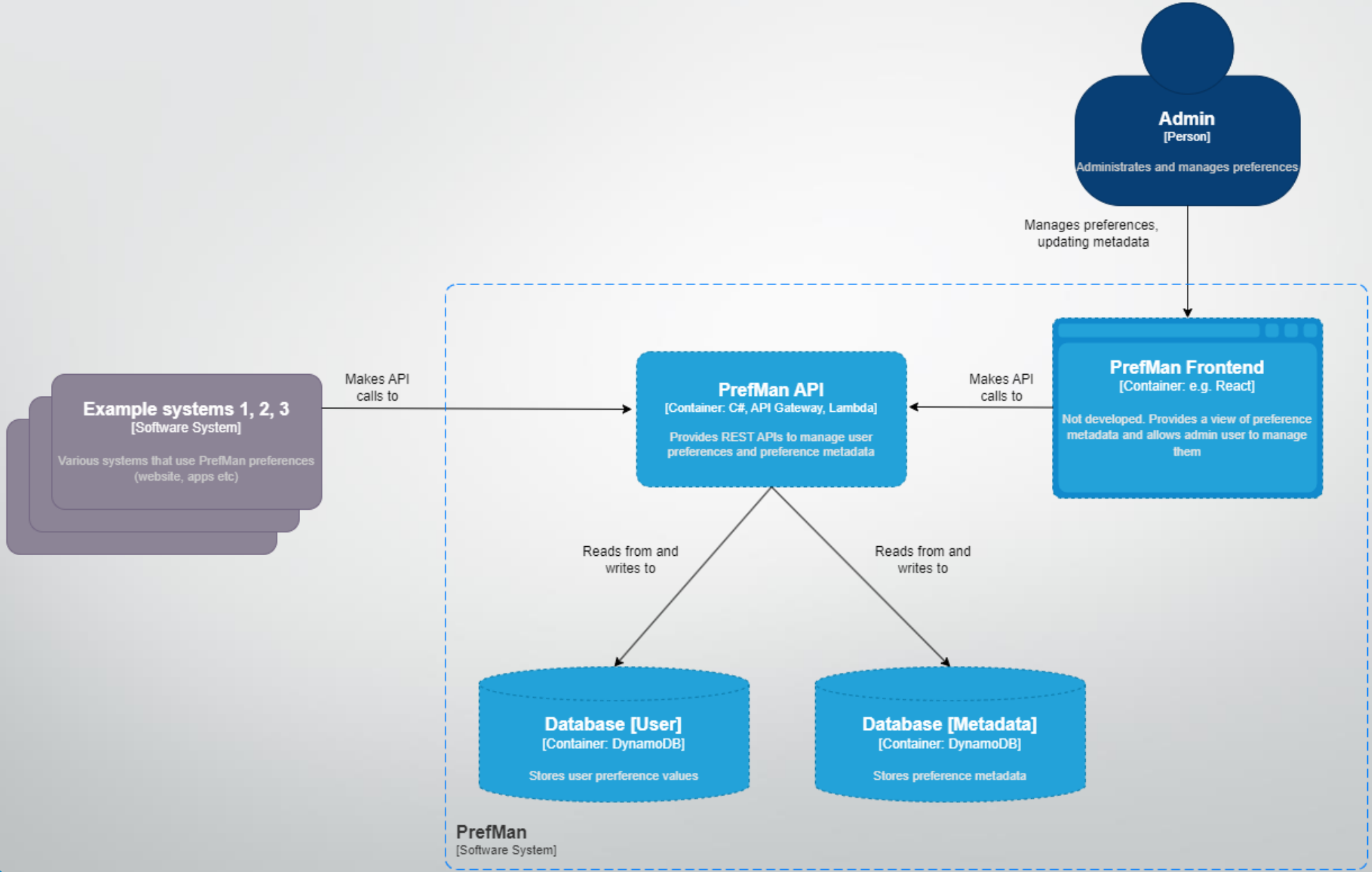
And the winner is…

- Three lambda functions
  - GetUserPreferences
  - PutUserPreferences
  - AdminPreferences
- C# functions, not using .NET Web API
- API Gateway
  - Triggers appropriate function based on route and method

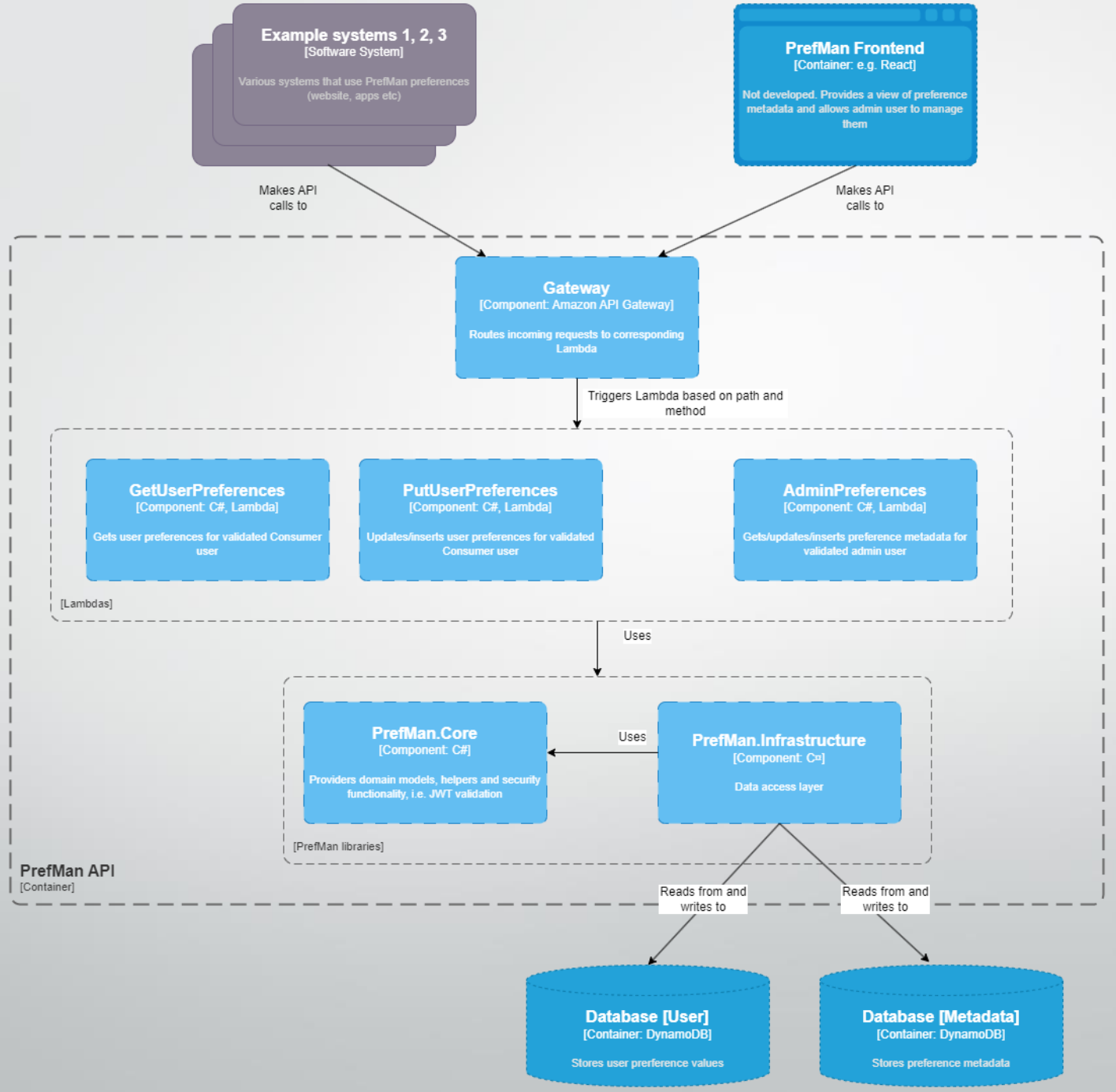
Let’s take a step back…



[System Context] PrefMan System

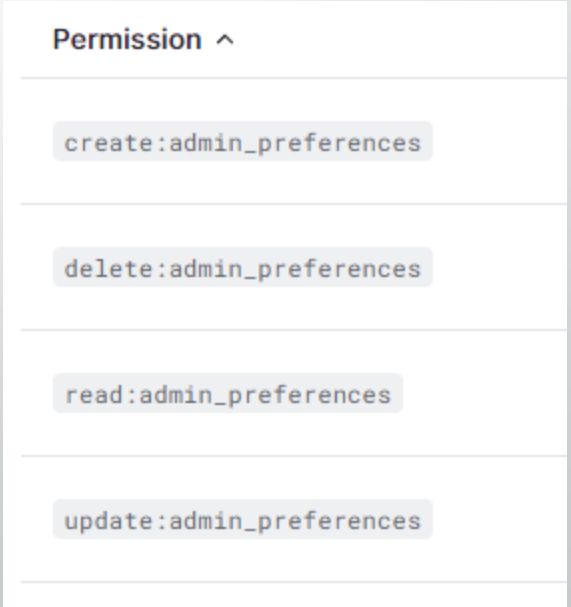


[Container] PrefMan System


Example systems 1, 2, 3
[Software System]
Various systems that use PrefMan preferences (website, apps etc)
PrefMan Frontend
[Container: e.g. React]
Not developed. Provides a view of preference metadata and allows admin user to manage them
Makes API calls to
Makes API calls to
Gateway
[Component: Amazon API Gateway]
Routes incoming requests to corresponding Lambda
Triggers Lambda based on path and method
GetUserPreferences
[Component: C#, Lambda]
Gets user preferences for validated Consumer user
PutUserPreferences
[Component: C#, Lambda]
Updates/inserts user preferences for validated Consumer user
AdminPreferences
[Component: C#, Lambda]
Gets/updates/inserts preference metadata for validated admin user
[Lambdas]
Uses
PrefMan.Core
[Component: C#]
Providers domain models, helpers and security functionality, i.e. JWT validation
Uses
PrefMan.Infrastructure
[Component: C¤]
Data access layer
[PrefMan libraries]
PrefMan API
[Container]
Reads from and writes to
Reads from and writes to
Database [User]
[Container: DynamoDB]
Stores user prerference values
Database [Metadata]
[Container: DynamoDB]
Stores preference metadata

Security

- Auth0 with RS256
- JWT is validated in each lambda using PrefMan.Core
  - Issuer, Audience, Lifetime, IssuerSigningKey are all validated
  - Could potentially be moved to a Lambda authorizer
- UserPreferences
  - The user ID (SID) is read from the payload after validating token
  - Consumers are only authorized to access/update resources for their own user ID
  - The user ID provided by Auth0 is used as primary key in DB
- AdminPreferences (metadata)
  - Permissions for an authenticated user is read from payload after validating token
  - CRUD operations each have their own permission
  - Currently, a user with the admin role has all 4 permissions



Infrastructure as Code

- The solution is configured to be set up with CDK
- CDK creates *PrefManStack* with:
  - The two Dynamo database tables
  - The three lambas
  - IAM roles with permissions for the lambdas
  - API Gateway with resources and methods
- No manual configuration/setup is needed with AWS Console/CLI
  - Except for 'aws configure' to setup credentials, and possibly 'cdk bootstrap' first time for account
- After running CDK, initial data can be seeded with PrefMan.Seeder

Limitations/areas of improvement

- Growing data model complexity, key-value store drawbacks
- Limited error handling and logging as of now
- Use of events/queues
  - Handle failovers better
  - Event to trigger user creation on first sign in
- Possibility to use GraphQL to query a subset of user preference values (not just 1 or all)
- Going from high level object persistance model to low-level API/Document model
  - Improve performance when interacting with DB (more fine-tuned querying, less serialization)
- Use/save appropriate types in DB (currently user preference values are always saved as strings)
- Switch to HTTP API for less overhead and added JWT features
  - To setup with CDK it seems to require Amazon.CDK.AWS.Apigatewayv2.Alpha which is experimental/alpha
- More automated tests, integration tests

Walkthrough of solution

- Code
- OpenAPI documenation
- Demo with Postman

Questions?