# 1.Introduktion

Detta dokument beskriver den tekniska implementationen av Växtvärk AB:s AI-baserade lösning för smart ogräsbekämpning. Lösningen är byggd i molnplattformen Amazon Web Services (AWS) och kombinerar flera tjänster för att leverera en interaktiv, datadriven och skalbar lösning.

### 1.1 Avvikelse från *Conceptual draft*

I *conceptual draft* fanns ursprungligen en ambition att inkludera autonoma robotar, drönare och lokala sensorer för att samla in realtidsdata från odlingsfält. I den faktiska implementationen har dessa komponenter valts bort. Denna förenkling har gjorts med hänsyn till teknisk komplexitet, kostnadseffektivitet samt för att möjliggöra en snabbare lansering. Det innebär också en förenkling av beslutsmatrisen från *Conceptual draft*, där parametern markfuktighet har uteslutits på grund av att sensordata inte ingår i den faktiska lösningen. En uppdaterad beslutsmatris finns i Appendix.

I övrigt levererar lösningen enligt den konceptuella designen och uppfyller de funktionella krav som specificerades i ett tidigare skede.

# 2. Översikt av AWS-arkitekturen

För att möjliggöra en flexibel, skalbar och robust arkitektur har lösningen implementerats med hjälp av Amazon Web Services (AWS). AWS tillhandahåller ett brett spektrum av molnbaserade tjänster som underlättar utveckling och drift av distribuerade system med hög tillgänglighet och låg driftskostnad.

Lösningen utnyttjar serverlösa teknologier såsom AWS Lambda för att hantera affärslogik, vilket minimerar behovet av att underhålla infrastruktur. Tjänster som Amazon Lex och Amazon Rekognition möjliggör integrerad användarinteraktion via naturligt språk och bildigenkänning, medan Amazon S3 används för datalagring:

**Kärnkomponenter:**

|  |  |
| --- | --- |
| Amazon Lex | Chatbot för användarinteraktion via text. |
| AWS Lambda | Serverlös backendlogik för att hantera intents och affärslogik. |
| Amazon Rekognition | Bildigenkänning för att artbestämma ogräs. |
| Amazon S3 | Lagring av JSON-data och ogräsbilder. |
| API Gateway | HTTP-gränssnitt för externa anrop från frontend. |
|  |  |

# 3. AWS-komponenter och teknisk uppsättning

Nedan följer en mer detaljerad beskrivning av funktionerna i AWS och hur de används.

### 3.1 Chatbot via Amazon Lex

Lex-boten har 6 så kallade intents:

* Getweedinfo – Hämtar information om ett specifikt ogräs (bildnamn som nyckel).
* GetHerbicideinfo-Hämtar information om ogräsmedel
* Weatherinfo – Hämta väderdata från SMHI för en plats och tidpunkt.
* CheckSprayConditions – Bedömer bekämpningsmöjlighet baserat på väder.
* ImageRecognition – Identifiering av ogräs genom bilduppladdning.
* Fallbackintent-Fångar upp meddelanden som inte matchar några andra intents och ger ett generiskt svar.

Varje intent i Amazon Lex definierar ett distinkt handlingsutrymme eller funktion som boten är konfigurerad att utföra, se ovan beskrivning för respektive intent. När användaren ger input analyserar Lex denna mot en uppsättning tränade utterances (exempelfraser) för att avgöra vilken intent som bäst matchar frågan. Exempel på en sån input är, *Vad blir det för väder i Göteborg?* som då kopplas till Weatherinfo. En intent fungerar alltså som en entry point till en specifik del av botens logik, vilket sedan triggar anrop till Lambda-funktionen för informationshämtning eller API-anrop.

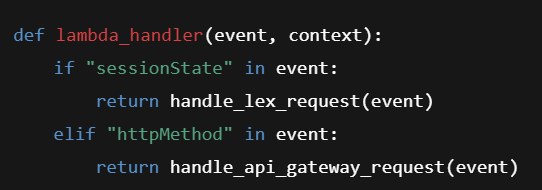
Vidare används *slots* för att extrahera nyckelinformation från användarens fråga – t.ex. *City* och *Date* i väderförfrågningar. Slots kan fyllas automatiskt baserat på det användaren skriver, eller genom uppföljningsfrågor från boten om informationen saknas.Lex använder även *sessionAttributes* (som definieras i Lambda) för att lagra information temporärt under en session, exempelvis senaste valda stad eller datum. Det gör att användaren kan ställa följdfrågor utan att behöva upprepa tidigare information.

Denna struktur gör det möjligt att bygga en naturlig och flexibel konversationsupplevelse, samtidigt som alla viktiga parametrar för vidare backend-logik fångas upp på ett kontrollerat sätt.

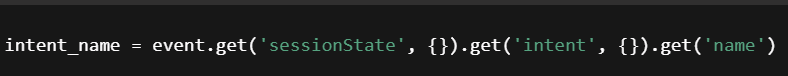
### 3.2 AWS Lambda – Backendlogik för Lex-intents

Lambdafunktionen är utvecklad i Python, vilket är ett naturligt val för detta användningsområde tack vare språkets utmärkta stöd för JSON-hantering, enkel integration med AWS SDK-software developement kit (boto3), samt tydlig och effektiv struktur för logik som svarar på event-drivna anrop. Python ger dessutom snabb prototypning och underlättar underhåll i en serverlös miljö.

Så AWS Lambda fungerar som backendmotor i lösningen och exekverar all affärslogik utan behov av dedikerade servrar. Funktionen är direkt integrerad med Amazon Lex och tar emot inkommande förfrågningar (events) när användaren interagerar med chatboten.Den centrala entry pointen är lambda\_handler, som avgör om anropet kommer från Lex eller från API Gateway:

****

Vid Lex-anrop tolkas vilken intent som använts med:

****

Baserat på detta aktiveras rätt logik, Weatherinfo för väderdata, Getweedinfo för ogräsinformation, GetHerbicideinfo för info om ogräsmedel eller CheckSprayConditions som är hårdkodad affärslogik för att utvärdera bekämpningsmöjlighet. Utöver interaktioner från Lex kan som sagt Lambda även ta emot externa HTTP-anrop via API Gateway, vilket möjliggör programmatisk åtkomst till samma backendlogik. Detta används vid platsbestämning/geokodning samt bildigenkänning.

### 3.3 Amazon S3 – Lagring av data och bilder för ogräsigenkänning

Amazon S3 (Simple Storage Service) är en skalbar objektlagringstjänst som används för att lagra och hämta data i form av objekt (t.ex. filer) i så kallade buckets. I den här lösningen fungerar S3 som central lagringsplats för både strukturerad och ostrukturerad data kopplat till ogräsigenkänning.

S3-bucketen innehåller tre huvudsakliga kataloger med olika syften:

* **OgräsDB/**: Här lagras två typer av resurser:
  + **JSON-filer** med strukturerad information om olika ogräsarter, inklusive latinskt namn, förekomst, giftighet, näringsvärde m.m.
  + **JPEG-bilder** kopplade till respektive ogräsart, vilka presenteras till användaren via chatboten.
* **Train/** och **Test/**: Dessa kataloger används i samband med träningsprocessen för bildigenkänningsmodellen i Amazon Rekognition Custom Labels. Här laddas bildmaterial upp manuellt för att bygga och validera AI-modellen för automatisk identifiering av ogräsarter.

S3 fungerar därmed som ett nav för både informationsförmedling till användare (via Lex och Lambda) och för maskininlärning i backend. Alla objekt är tillgängliga för Lambda-funktionen via boto3 och hämtas dynamiskt beroende på användarens fråga.

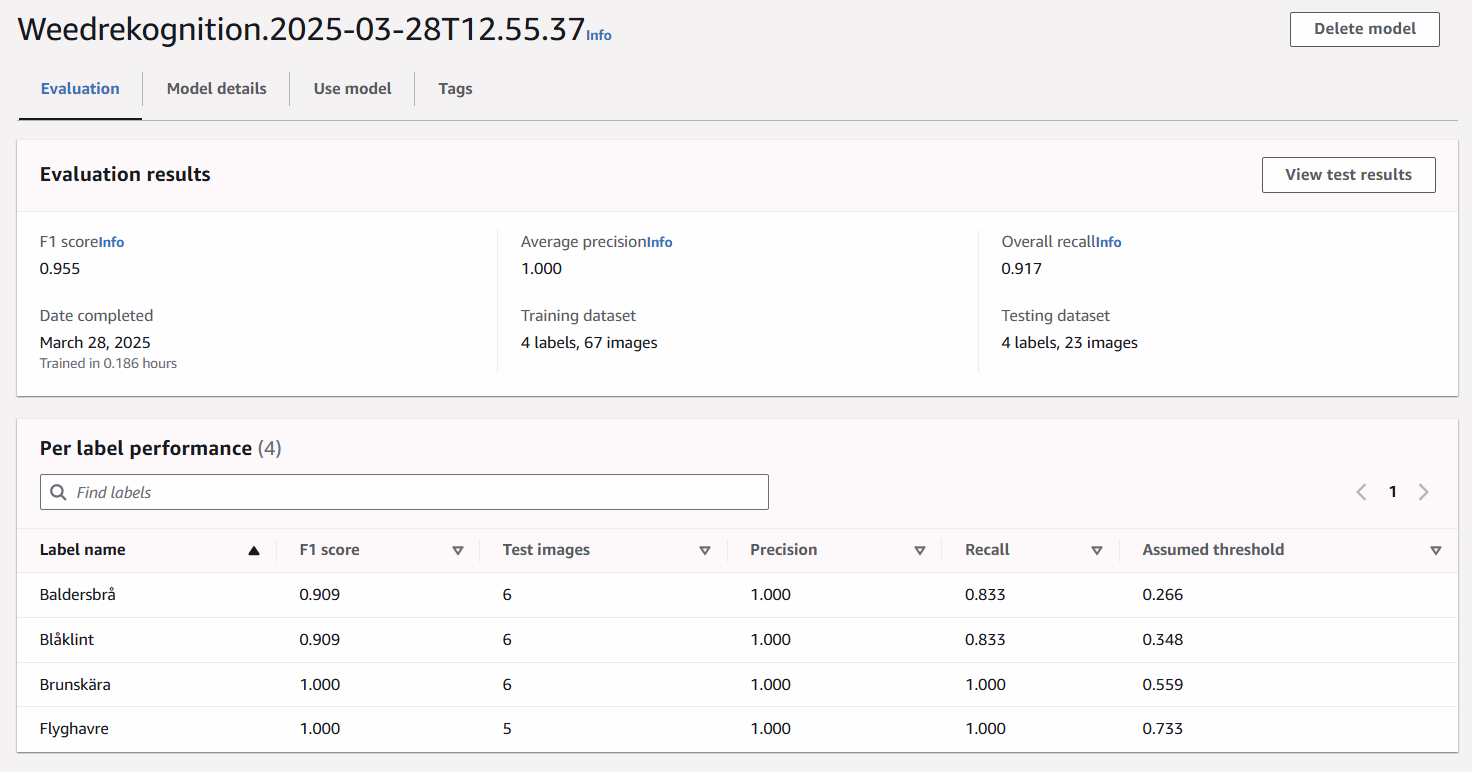
### 3.4 Amazon Rekognition – bildigenkänning

Amazon Rekognition används för att identifiera ogräsarter baserat på uppladdade bilder från Internet. Det är en maskininlärningstjänst från AWS för bildanalys som gör det möjligt att skapa och träna egna modeller för att känna igen specifika objekt, alltså artbestämma ogräs.

I implementationen har en *Custom Labels-modell* tränats på ett begränsat dataset bestående av totalt 90 bilder för fyra ogräsarter:

* Balderbrå
* Blåklint
* Brunskära
* Flyghavre

Modellen tränades direkt i AWS:s användargränssnitt och visar redan med begränsat dataset goda resultat med ett totalt F1-score på 0.955. Det innebär en välbalanserad precision och recall, trots det få antal tränings- och testbilder. Exempelvis har Brunskära och Flyghavre 100% precision och recall i testsetet.



När en användare laddar upp en bild i gränssnittet skickas den till en Lambda-funktion som i sin tur anropar modellen via boto3, där det mest sannolika ogräset identifieras och visas för användaren tillsammans med en sannolikhetsbedömning.

Detta tillvägagångssätt möjliggör ett kraftfullt och automatiserat sätt att tolka bilder och kan enkelt skalas upp genom att addera fler bilder eller nya ogräsarter.

### 3.5 API Gateway

API Gateway fungerar som en exponeringsyta för Lambda-funktioner via REST-API:er och möjliggör integrering med externa system. I lösningen används API Gateway för att skicka bilder till bildigenkänning (/identify) och för att göra manuella anrop till vädermodulen. Anropen stödjer POST och OPTIONS med CORS-konfiguration, vilket gör det möjligt att använda tjänsterna från webbaserade klienter eller externa applikationer på ett säkert sätt.

# 4. Övriga komponenter

Utöver AWS-tjänsterna innehåller lösningen även andra nyckelkomponenter som stödjer användargränssnitt, geokodning och väderdata. Användarinteraktionen sker via ett webbgränssnitt byggt i Streamlit, där dialog med användaren hanteras samt bilder kan laddas upp. För att omvandla ortsnamn till koordinater används OpenCage API, och aktuell väderinformation hämtas från SMHI:s öppna väder-API. Dessa komponenter samverkar med AWS-baserad logik för att skapa en helhetslösning för smart ogräsbekämpning.

### 4.1 Streamlit

För att möjliggöra en användarvänlig och visuell interaktion har en frontend utvecklats med hjälp av *Streamlit*, ett Python-baserat ramverk för snabb prototypning av webbgränssnitt. Applikationen fungerar som det primära användargränssnittet och integrerar direkt med Amazon Lex för chattbaserad dialog och API Gateway för bildidentifiering.

Applikationen består av två huvudsakliga sektioner:

* **Chattgränssnitt mot Lex**:  
  Användaren kan skriva naturliga frågor om väder eller bekämpningsmöjligheter. Dessa frågor skickas till Lex via boto3-klienten (recognize\_text) och svaren presenteras med chattbubblor, inklusive stöd för emojis. Sessionen hanteras lokalt i st.session\_state och upprätthåller kontexten för dialogen.
* **Bilder för ogräsidentifiering**:  
  Genom ett expanderbart avsnitt kan användaren ladda upp en bild (JPG/JPEG) som skickas till en serverlös bildigenkänningsfunktion via en POST-förfrågan till en API Gateway-endpoint (/identify). Bilden kodas i base64 och skickas med requests, varefter svaret (identifierat ogräs) visas direkt i gränssnittet.

Dessutom innehåller appen möjlighet att rensa dialoghistorik och inaktivera sessionen hos Lex. All logik är tydligt separerad i sektioner för UI, inputhantering och API-integration. Layouten har anpassats med anpassad CSS för att efterlikna en modern chattupplevelse.

### 4.2 Geokodning via OpenCage

För att hämta väderdata krävs latitud och longitud. Detta uppnås genom att konvertera ortsnamn till koordinater via en extern geokodningstjänst (OpenCage API). När en användare anger en plats i chatten anropas OpenCage automatiskt från Lambda, vilket returnerar exakta koordinater som används i nästa steg.

### 4.3 Väderdata från SMHI

Lösningen hämtar väderdata från SMHI:s öppna API (<https://opendata-download-metfcst.smhi.se/>) för att kunna ge användaren aktuell väderinformation. Anropen sker mot punktprognoser baserat på longitud och latitud.

I varje anrop efterfrågas en specifik prognos för kl. 10:00 UTC samma dag som användaren angett. Prognoserna innehåller flera tidpunkter per dag, och logiken i systemet identifierar automatiskt den datapunkt som ligger närmast kl. 10. Detta säkerställer att användaren får ett jämförbart väderutslag för varje förfrågan.

Exempel på efterfrågade parametrar:

* Temperatur
* Vindhastighet och byvind
* Luftfuktighet
* Nederbörd
* Molntäcke
* Vindriktning

# 5. Säkerhetsaspekter

I detta projekt har säkerhet inte varit ett primärt fokusområde, utan lösningen har medvetet konfigurerats med generösa behörigheter för att möjliggöra snabb prototypning och testning. Detta inkluderar breda IAM-policyer (bland annat AmazonRekognitionFullAccess och AmazonS3FullAccess), samt egna inline-policys för åtkomst till Lex-sessioner och bildigenkänning.

Vidare är S3-bucketen som används (ulbjogras) konfigurerad med publik åtkomst aktiverad, vilket förenklar integrationen med externa verktyg och webbtjänster, men innebär en säkerhetsrisk i ett produktionsscenario.

Framöver, för en säker och hållbar produktionsmiljö, bör följande åtgärder beaktas:

* Begränsa IAM-behörigheter enligt principen om minsta åtkomst ("least privilege").
* Stäng av publik åtkomst till S3 och använd signerade URL:er vid behov.
* Inför API-nycklar eller annan autentisering vid API Gateway.
* Aktivera loggning och övervakning (exempelvis via CloudTrail) för att spåra åtkomst till resurser.

Syftet med nuvarande uppsättning har varit att skapa en funktionell helhet med kort utvecklingstid. Säkerheten ses därmed som ett område för framtida förstärkning vid vidareutveckling av lösningen.

# 6. Tekniska Förbättringar – Nästa steg

Även om lösningen är fullt fungerande i sitt nuvarande skick, finns ett antal tekniska förbättringsåtgärder som skulle kunna öka både skalbarhet, säkerhet och funktionalitet i ett vidareutvecklat system.

* **DynamoDB-integration:** Istället för att lagra ogräsinformation som statiska JSON-filer i S3, kan informationen struktureras och lagras i en DynamoDB-tabell. Det möjliggör mer flexibel sökning, uppdatering och hantering av metadata kring ogräsarter – samt framtida utökningar som versionering eller relationsdata.
* **Automatisk bildklassificering:** I nuläget sker identifiering av ogräs manuellt via en knapptryckning i gränssnittet. En mer automatiserad lösning skulle vara att direkt analysera nya bilder som laddas upp till S3 med hjälp av en Lambda-trigger som aktiverar Amazon Rekognition. Detta skulle på sikt möjliggöra kontinuerlig datainsamling från exempelvis fältkameror.
* **IoT-data:** Systemet är designat för att potentiellt kunna integrera realtidsdata från sensorer ute i fält. Genom att exponera API Gateway-endpoints kan externa IoT-enheter skicka data (såsom luftfuktighet, temperatur, vindstyrka) som används i bekämpningsanalysen. Detta öppnar upp för helt automatiserade beslutsstöd.
* **Dashboard och visualisering:** Ett naturligt nästa steg är att bygga ett mer omfattande gränssnitt – exempelvis i Streamlit eller med hjälp av Amazon QuickSight – där användaren får en överblick över historik, status på identifieringar, samt statistik över väderförhållanden och bekämpningslägen.
* **IAM-förfining:** I prototypen har breda behörigheter använts. Ett viktigt förbättringssteg är att införa mer finkornig åtkomstkontroll. Detta innebär att olika delar av applikationen får endast den minimala åtkomst som krävs, exempelvis genom uppdelning i olika IAM-roller och principer för Lambda, API Gateway och klienten.

Dessa förbättringar representerar olika vägar att skala upp systemet mot en mer robust, produktionsklar lösning. Flera av dem kan dessutom implementeras stegvis, vilket gör det möjligt att successivt höja värdet och funktionaliteten i tjänsten.

# Appendix

Uppdaterad beslutsmatris för att avgöra huruvida ogräsbekämpning är att rekommendera.

