DVS Technologies

Generative AI & Agentic AI Course Content

INTRODUCTION TO GENERATIVE AI AND AGENTIC AI

- Artificial Intelligence, Machine Learning, Deep Learning and Generative AI fundamentals
- Large Language Models, Prompt Engineering, RAG, vector databases and AI agents
- Python programming and AWS cloud services for application development
- LangChain, Neo4j, LangGraph, Docker, Kubernetes, MCP, Bedrock, Bedrock Agent Core and Amazon EKS
- Hands-on labs, mini-projects, interview preparation and end-to-end projects

MODULE 1: GENERATIVE AI

OVERVIEW

This module introduces the foundations of AI, Machine Learning, Generative AI and Large Language Models. Students learn how modern language models work, how conversations are managed, and how token pricing and context windows affect application design.

1. INTRODUCTION TO AI AND MACHINE LEARNING

- Artificial Intelligence, Machine Learning and Deep Learning concepts
- Supervised, unsupervised and reinforcement learning overview
- Traditional programming versus machine-learning systems
- Common business applications of AI and ML

2. INTRODUCTION TO GENERATIVE AI

- What Generative AI is and how it differs from predictive AI
- Text, code, image, audio and multimodal generation
- Generative AI use cases across business functions
- Benefits, limitations and responsible use

3. TIMELINE OF GENERATIVE AI

- Evolution from rule-based systems to Machine Learning and Deep Learning
- Word embeddings, sequence models and attention
- Transformer models, foundation models and conversational AI
- Emergence of AI agents and agentic workflows

4. GENERATIVE AI LANDSCAPE

- Foundation-model providers and open-weight models
- LLM application frameworks and orchestration tools
- Vector databases, search systems and knowledge sources
- Evaluation, observability, security and deployment platforms

5. INTRODUCTION TO LARGE LANGUAGE MODELS

- Foundation models, language models and model parameters
- Pretraining, instruction tuning and inference
- Base models, chat models and reasoning models
- Model selection based on quality, latency, cost and privacy

6. LLM ARCHITECTURE

- Tokenization, embeddings and positional information
- Transformer and self-attention concepts
- Feed-forward layers and next-token prediction
- End-to-end request and response flow

7. CONVERSATION HISTORY

- System, user, assistant and tool messages
- Stateless and stateful conversations
- Conversation buffers, summaries and sliding windows

- Short-term memory, long-term memory and privacy considerations

8. TOKEN PRICING

- Input, output and cached tokens
- Token-count and cost estimation
- Factors affecting application cost and latency
- Prompt, context and model optimization techniques

9. CONTEXT WINDOW

- Meaning of model context window and output limits
- Context overflow and lost-in-the-middle problems
- Context selection, compression and summarization
- Using retrieval to provide relevant external context

10. PROMPT ENGINEERING VERSUS CONTEXT ENGINEERING

- Prompt instructions, constraints, examples and output format
- Context from documents, memory, tools, databases and user state
- Difference between writing a prompt and designing a context pipeline
- Enterprise examples of prompt and context engineering

MODULE 2: AWS CLOUD

OVERVIEW

This module covers the AWS services required to host Python and Generative AI applications. Students create secure identities, launch compute resources, work with storage and databases, build containers, and expose serverless applications through APIs.

AWS ACCOUNT CREATION

- Create an AWS account and understand regions and Availability Zones
- Secure the root account with multi-factor authentication
- Review billing, Free Tier usage, budgets and cost alerts
- Navigate the AWS Management Console

1. IDENTITY AND ACCESS MANAGEMENT

Creating an IAM User without Programmatic Access

- Create a console-only IAM user
- Configure password policy and multi-factor authentication
- Avoid daily use of the AWS root account

Assigning Permissions to an IAM User

- AWS-managed, customer-managed and inline policies
- Policy elements: effect, action, resource and condition
- Apply the principle of least privilege

Working with Groups

- Create role-based IAM groups
- Attach policies to groups and add users
- Manage permissions consistently across teams

Configure Service Account and AWS CLI

- Install and configure the AWS CLI
- Create named profiles and verify caller identity
- Securely handle access keys and environment variables

Working with Roles

- Understand trust policies and permission policies
- Attach IAM roles to EC2 and Lambda
- Use temporary credentials for AWS service access

2. ELASTIC COMPUTE CLOUD

EC2 Instance Creation and Login

- Select an AMI, instance type, key pair, storage and network settings
- Configure security-group rules
- Connect using SSH or EC2 Instance Connect
- Install packages and run a basic application

3. SIMPLE STORAGE SERVICE

Create S3 Bucket and Access Data

- Create buckets and upload, download and organize objects
- Access S3 through the console, CLI and Python
- Configure IAM permissions, encryption and versioning

S3 Architecture

- Buckets, objects, keys, prefixes and metadata
- Regional architecture, durability and availability
- Bucket policies, lifecycle rules and presigned URLs

4. RELATIONAL DATABASE SERVICE

RDS Instance Creation and Connection

- Create a PostgreSQL RDS database
- Configure subnet, security group and connectivity
- Connect from a database client and Python application
- Understand backups, monitoring and credential security

5. DOCKER AND AMAZON ELASTIC CONTAINER REGISTRY

Running a Python Application on EC2

- Install Python and create a virtual environment
- Install dependencies and configure environment variables
- Run and troubleshoot the application on EC2

Working with Docker

- Images, containers, Dockerfile and container lifecycle
- Build and run a Python application image
- Ports, environment variables, volumes and networks

Working with Amazon ECR

- Create an ECR repository
- Authenticate Docker with ECR
- Tag, push, pull and version container images

Docker and ECR Integration

- Build the application image
- Push the image to ECR
- Run the ECR image on EC2 or another AWS service

6. AWS LAMBDA

Working with Lambda

- Serverless concepts, handlers, events and context
- Execution roles, environment variables, memory and timeout
- Testing, logs and monitoring through CloudWatch

Working with Lambda Container Images

- Prepare a Lambda-compatible Docker image
- Push the image to Amazon ECR
- Create and invoke a Lambda function from the image

7. AMAZON API GATEWAY

- REST API and HTTP API concepts
- Resources, methods, routes and Lambda proxy integration
- Path parameters, query parameters, request body and responses
- Deployment stages, CORS, logging and basic security

PROJECT 1: Serverless Employee Management REST API Using AWS Lambda, API Gateway, and DynamoDB

Project Description

Designed and developed a serverless REST API application using AWS Lambda, Amazon API Gateway, and Amazon DynamoDB to perform employee management operations. The

application provides CRUD capabilities to create, read, update, and delete employee records through REST API endpoints.

The solution follows a serverless architecture, where API Gateway receives client requests and triggers AWS Lambda functions to process business logic. Lambda integrates with DynamoDB to store and retrieve employee information with high scalability, availability, and low operational overhead.

The application supports both single and bulk employee record creation, allowing users to upload multiple employee details in a single API request. DynamoDB batch operations are implemented to efficiently process multiple records.

AWS & Other Services Used

MODULE 3: PYTHON

OVERVIEW

This module develops the Python skills required for cloud, data and Generative AI application development. It covers core syntax, collections, functions, modules, files, object-oriented programming, validation, logging and projects.

1. PYTHON DOMAIN AND PLATFORM INTRODUCTION

- Python use cases in automation, web, data, cloud and AI
- Python installation, IDEs, notebooks and virtual environments

2. PYTHON BASICS

- Syntax, indentation, comments, input and output
- Variables, data types, conversions and operators

3. LOOPS, COLLECTIONS AND FUNCTIONS

- Conditions, for loops and while loops
- Lists, sets, tuples, dictionaries and reusable functions

4. PYTHON FUNCTIONS

- Parameters, arguments, return values and scope
- Type hints, docstrings and reusable design

5. MODULES, PACKAGES AND IMPORTS

- Import and from-import statements
- Create reusable modules, packages and requirements files

6. PYTHON MINI-PROJECT

- Build a menu-driven student or employee management application
- Apply functions, collections and validation

7. PYTHON FILE OPERATIONS

- Read, write and append files
- Work with text, JSON and CSV data

8. OBJECT-ORIENTED PROGRAMMING IN PYTHON

- Classes, objects, attributes, methods and constructors
- Instance, class and static methods

9. PYTHON INHERITANCE

- Parent and child classes
- Method overriding and super()

10. PYTHON POLYMORPHISM AND ENCAPSULATION

- Common interfaces, overriding and duck typing
- Public, protected and private members

11. PYTHON ABSTRACTION

- Abstract base classes and abstract methods
- Separate interface from implementation

12. PYTHON OOP MINI-PROJECT

- Develop an object-oriented application
- Apply inheritance, polymorphism, encapsulation and abstraction

13. PYTHON EXCEPTION HANDLING

- try, except, else and finally
- Raise built-in and custom exceptions

14. PYTHON LOGGING

- Logging levels, formatting and handlers
- Console, file and exception logging

15. PYTHON PYDANTIC DATA VALIDATION

- Create models and validate structured input
- Fields, nested models, validators and validation errors

16. PYTHON BANKING PROJECT

- Customer, account and transaction operations
- OOP, validation, custom exceptions and logging

PYTHON AND AWS FINAL PROJECT

1. Python Project Services

- Python, AWS Lambda, API Gateway, PostgreSQL RDS, IAM and CloudWatch
- Pydantic validation and optional Docker/ECR packaging

2. Python Project Architecture

- Client to API Gateway to Lambda to PostgreSQL request flow
- Service, validation, database, logging and exception layers

3. Python Project Implementation

- Implement create, read, update and transaction operations
- Add configuration, validation, errors, logs and secure database access

4. Understanding the Code

- Explain folder structure, modules and application entry point
- Trace request processing, database operations and responses

PROJECT: BANKING MINI PROJECT

- Create customers and accounts
- Deposit, withdraw and check balances
- Maintain transaction history
- Use OOP, Pydantic, custom exceptions, file or database storage, and logging

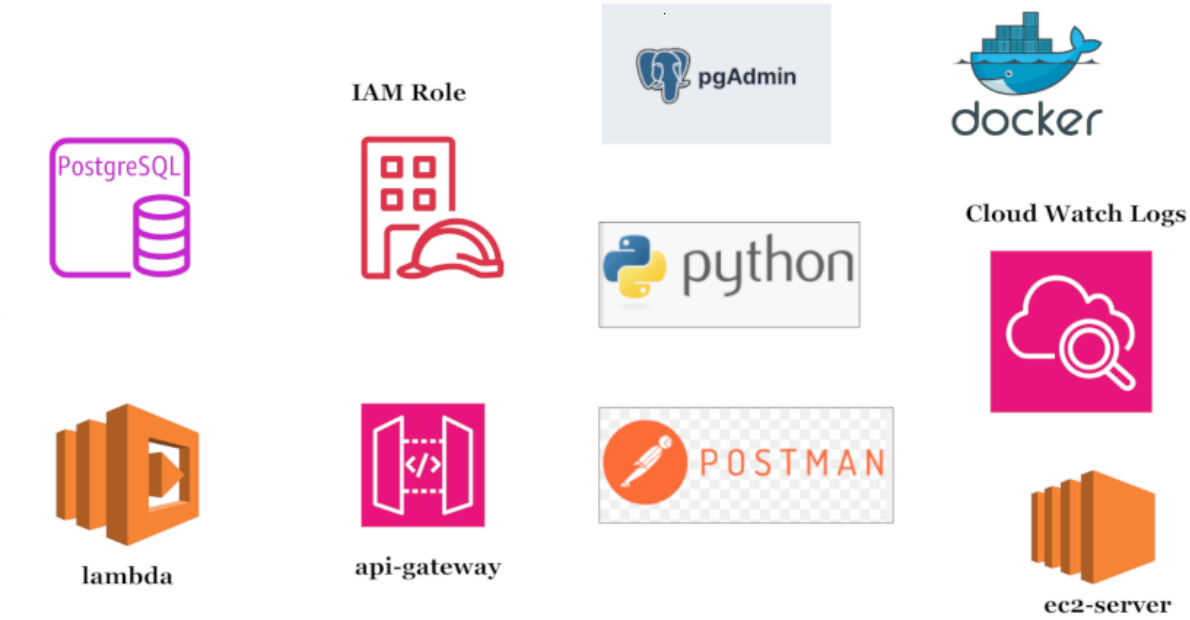
PROJECT 2: AWS Serverless Customer and Order Management API using AWS Lambda, Amazon API Gateway, Amazon RDS PostgreSQL, Amazon ECR, AWS Secrets Manager, AWS IAM, Amazon CloudWatch and Amazon EC2

Project Description

Develop a containerized serverless REST API for managing customer and order information using Python and AWS. Amazon API Gateway exposes CRUD endpoints, while separate AWS Lambda functions handle customer and order operations. A reusable **Generic Query Executor Lambda** securely connects to an Amazon RDS PostgreSQL database and executes validated SQL queries.

Docker images are stored in Amazon ECR, database credentials are managed through AWS Secrets Manager, service permissions are controlled using IAM roles, and application logs are monitored through Amazon CloudWatch. The project also implements request validation, customer–order relationship checks, structured responses, exception handling, and database integrity.

AWS & Other Services Used



MODULE 4: PROMPT ENGINEERING

OVERVIEW

This module explains how language models process text and how prompts can be designed for reliable business tasks. Students create model and search accounts, experiment with prompting methods, tune model parameters and build a Text-to-SQL project.

CREATE OPENAI AND TAVILY ACCOUNTS

- Create accounts and generate API keys
- Configure environment variables and .env files
- Set usage limits and protect secrets from source control

1. LLM ARCHITECTURE AND BYTE PAIR ENCODING

1. What Is an LLM?

- Language models, foundation models and parameters
- Pretraining, instruction tuning and inference

2. LLM Problems

- Hallucination, outdated knowledge, bias and context limitations
- Cost, latency, privacy and non-deterministic output

3. Enterprise LLM Architecture

- User interface, API, model gateway and LLM
- Knowledge sources, tools, security, observability and evaluation

4. How an LLM Works

- Text to tokens to embeddings to transformer layers
- Attention, probability calculation and next-token generation

5. Byte Pair Encoding

- Words, subwords and vocabulary
- Frequent-pair merging and tokenization examples
- Impact of tokenization on context and pricing

2. PROMPT ENGINEERING

1. Prompt Engineering Fundamentals

- Instructions, context, input, constraints and output format
- Clear, specific and testable prompts

2. Different Types of Prompting

- Zero-shot, one-shot and few-shot prompting
- Role, structured-output, ReAct and tool-use prompting

3. Practical Prompt Examples

- Customer support, extraction, classification and content generation
- Code, SQL, summarization and data-transformation use cases

4. Prompt Development Basics

- Create initial prompt and evaluate the response
- Add examples, constraints and explicit success criteria

5. Iterative Prompt Development

- Test, identify failures, refine and version prompts
- Compare outputs across models and parameters

6. Summarization

- Concise, detailed, executive and audience-specific summaries
- Control length, format and information coverage

7. Inferring

- Sentiment, intent, urgency, topics and entity extraction
- Classification with structured output

8. Transformation

- Translation, tone change, correction and format conversion
- Transform unstructured content into structured data

9. Expanding

- Expand notes into emails, reports and proposals
- Generate controlled content using supplied facts

10. Model Parameters

- Temperature, maximum output tokens and top-p
- Frequency penalty, presence penalty and stop conditions

PROJECT 3: GenAI-Powered Text-to-SQL Customer and Order Assistant

Project Description

Build a Generative AI application that allows users to query customer and order information using natural-language questions instead of manually calling multiple APIs or writing SQL queries.

The application uses Prompt Engineering and an LLM to convert the user's question into a validated PostgreSQL query. A Generic Query Executor securely executes the query against Amazon RDS PostgreSQL, and the database results are converted into a clear, human-readable response.

The solution uses AWS Lambda, API Gateway, Amazon RDS, Docker, Amazon ECR, AWS Secrets Manager, IAM and CloudWatch to provide a secure, serverless and containerized GenAI architecture.

Services Used

MODULE 5: RETRIEVAL-AUGMENTED GENERATION

OVERVIEW

This module teaches how to connect language models with private and current data. Students build ingestion and retrieval pipelines, work with Pinecone, compare chunking strategies, and optimize retrieval with hybrid search and reranking.

1. RETRIEVAL-AUGMENTED GENERATION

1. The RAG Problem

- LLMs do not automatically know private or frequently changing data
- Long documents, hallucination and source-grounding challenges

2. What Is RAG?

- Retrieve relevant knowledge and add it to the prompt
- Generate evidence-based responses with sources

3. Traditional Database versus Vector Database

- Exact and structured search versus semantic similarity search
- Rows and keys versus vectors and metadata

4. Ingestion

- Load, clean, split and enrich documents
- Create embeddings and store vectors with metadata

5. Retrieval

- Embed the question and search for relevant chunks
- Top-k, filters, similarity scores and reranking

6. Augmented Generation

- Construct a grounded prompt from retrieved context
- Generate answers, citations and insufficient-evidence responses

7. Final RAG Architecture

- Document store, ingestion service, embedding model and vector database
- Retriever, reranker, prompt, LLM, evaluation and monitoring

8. File-to-Vector-Database Calculation

- Estimate characters, tokens, chunks and vector records
- Estimate embedding, storage and query cost

2. PINECONE RAG

1. Vector Database Types

- Local, in-memory, managed and cloud-native vector stores
- Selection based on scale, latency, operations and cost

2. Working with Pinecone

- Create an index, select dimensions and similarity metric
- Upsert, query, filter, update and delete vectors

3. Working with Hugging Face Datasets

- Load and inspect datasets
- Clean records, generate embeddings and upload vectors

4. Pinecone Simple RAG

- Load documents, chunk text and create embeddings
- Retrieve context and generate a grounded answer

5. Tips and Recommendations

- Use meaningful metadata and version documents
- Tune chunk size, top-k and retrieval filters

6. Two-Stage RAG

- Retrieve a broad candidate set
- Rerank candidates before sending context to the model

7. Need for Chunking Strategies

- Balance semantic completeness, retrieval precision and cost
- Avoid chunks that are too small or too large

8. Chunking Strategies

- Fixed, recursive, sentence, paragraph and semantic chunking
- Markdown-aware and parent-child chunking

9. Context-Aware Chunking

- Include titles, sections, summaries and neighboring context
- Improve chunk meaning before embedding

10. Hybrid Search Optimization

- Combine dense semantic search with sparse keyword search
- Use BM25, score fusion, metadata filters and reranking

PROJECT 4: Enterprise HR Policy RAG Assistant

Project Description

Build a Generative AI application that helps employees retrieve accurate answers from company HR documents using natural-language questions.

HR policies such as leave management, attendance, work-from-home, onboarding, employee benefits, travel reimbursement, code of conduct and resignation procedures are uploaded and processed through a RAG pipeline. The documents are divided into chunks, converted into embeddings and stored in a vector database.

When an employee asks a question, the application retrieves the most relevant policy content and sends it to the LLM as context. The LLM generates a grounded response with the source document and policy section.

Services Used

Python, Amazon S3, Amazon ECS or Lambda, OpenAI, Pinecone, Docker, Amazon ECR, AWS Secrets Manager, IAM and CloudWatch.

MODULE 6: LANGCHAIN

OVERVIEW

This module introduces LangChain as a framework for building reusable LLM applications. Students work with prompts, models, parsers, chains, retrievers and query-translation techniques to create advanced RAG workflows.

1. WHY LANGCHAIN?

- Reusable abstractions for models, prompts, parsers and retrievers
- Simplify chaining, tool integration, streaming and application composition

2. WORKING WITH LANGCHAIN

- Chat models, prompt templates, messages and output parsers
- LangChain Expression Language, sequential and parallel runnables

3. LANGCHAIN AGENTIC RAG AND HYBRID SEARCH

- Route questions to vector, keyword, SQL or web-search tools
- Combine retrieval, tool results and source-grounded generation

4. QUERY TRANSLATION TECHNIQUES

- Query rewriting, multi-query retrieval and query expansion
- RAG Fusion, reciprocal rank fusion, decomposition and step-back prompting

MODULE 7: GRAPH DATABASE AND NEO4J

OVERVIEW

This module introduces graph databases for highly connected enterprise data. Students learn the Neo4j property-graph model, write Cypher queries and build an airline knowledge-graph chatbot.

1. INTRODUCTION TO GRAPH DATABASE AND NEO4J

1. Why Graph Databases?

- Model relationship-heavy data without complex joins
- Fraud, recommendations, networks, dependencies and knowledge graphs

2. What Is Neo4j and Cypher?

- Neo4j property-graph database

- Cypher pattern-matching query language

3. Core Components of Neo4j

- Nodes, labels, relationships and properties
- Paths, patterns, indexes and constraints

4. Graphs Are Everywhere

- Airline routes, banking transactions and social networks
- Application dependencies, supply chains and enterprise knowledge

2. WORKING WITH CYPHER

1. Lab Setup

- Create a Neo4j database and open Neo4j Browser
- Connect to Neo4j from Python

2. Reading Data

- MATCH, RETURN, aliases, sorting and limiting
- Read nodes, relationships and properties

3. Finding Relationships

- Direction, relationship types and multi-hop traversal
- Paths and variable-length relationships

4. Filtering Queries

- WHERE, comparisons, string filters and null handling
- Aggregations, grouping and metadata conditions

PROJECT5: GenAI-Powered Airline Travel Knowledge Graph Chatbot

Project Description

Build an intelligent travel assistant that allows users to query airline, flight, hotel, city, booking and passenger information using natural language. The project generates realistic travel data and stores it in Neo4j as connected nodes and relationships.

Using Python, LangChain, an LLM and Neo4j Cypher, the application converts user questions into graph queries, retrieves connected information from Neo4j and returns conversational answers through a Chainlit interface.

Services Used

MODULE 8: LANGGRAPH

OVERVIEW

This module teaches stateful and controllable agentic workflows using LangGraph. Students create graphs, routers and agents, then add memory and durable PostgreSQL persistence.

1. WHY LANGGRAPH?

- State management, branching, loops and human approval
- Reliable control for enterprise agent workflows

2. SIMPLE GRAPH

- Define state, nodes, edges, start and end
- Compile, invoke and visualize a graph

3. LANGSMITH STUDIO

- Trace runs, inspect prompts, tool calls, tokens and latency
- Debug workflows and evaluate outputs

4. CHAINS

- Build sequential workflows with shared state
- Pass and transform values between nodes

5. ROUTER

- Classify requests and use conditional edges
- Route to LLM, retrieval, SQL or tools

6. AGENT

- Define tools and execute agent-tool loops
- Control exit conditions, iterations and failures

7. AGENT WITH MEMORY

- Maintain message history across follow-up questions
- Use thread identifiers and conversation state

8. PERSISTENCE

- Checkpoints, snapshots, replay and state updates
- Resume interrupted or failed workflows

9. POSTGRESQL PERSISTENCE

- Store checkpoints and thread state in PostgreSQL
- Support durable, multi-session agent conversations

PROJECT 6: AI-Powered Travel Planning and Booking Agent using LangGraph on Amazon EKS

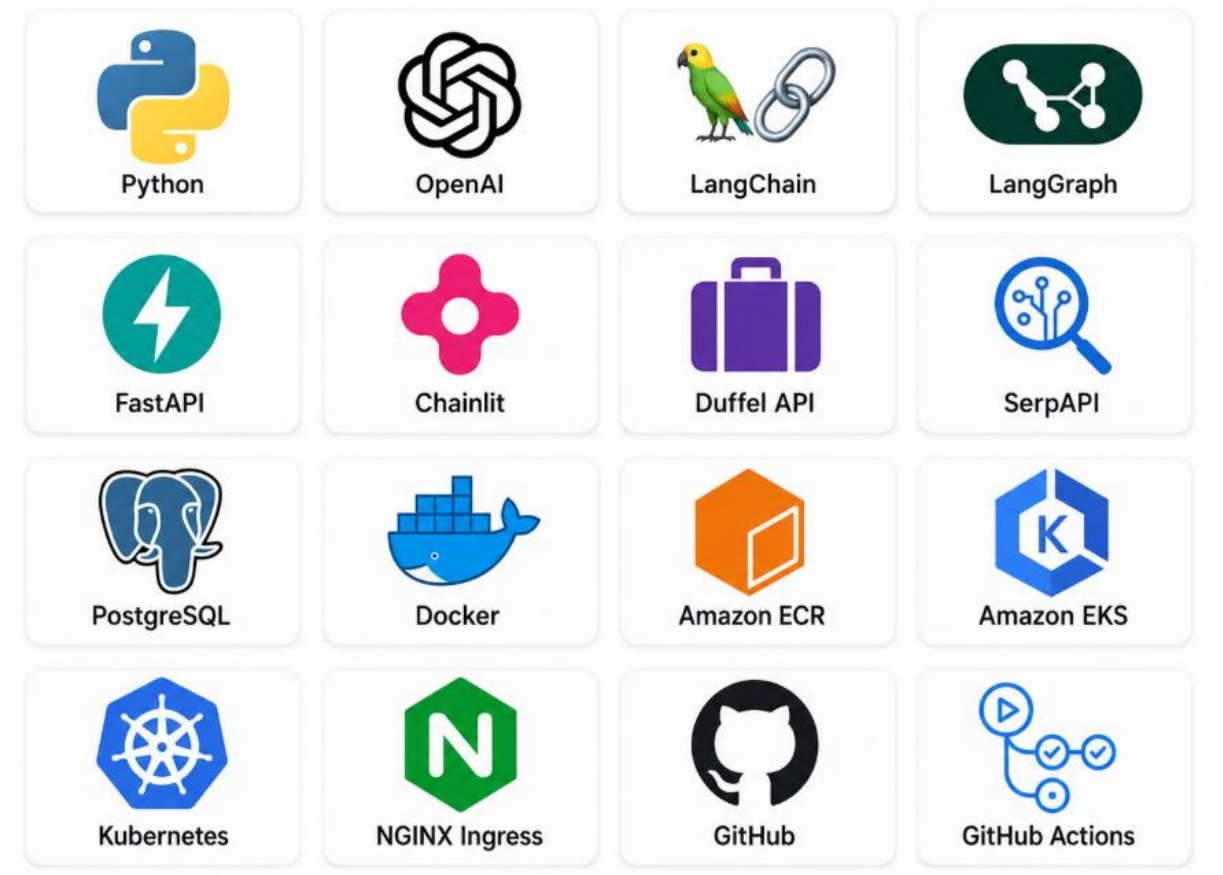
Project Description

Develop an enterprise-grade Agentic AI travel assistant that creates personalized travel plans based on the user's source, destination, travel date and budget. The Chainlit user interface communicates with a FastAPI and LangGraph backend, which manages the complete travel workflow, including airport-code identification, live flight search using Duffel, hotel and sightseeing search using SerpAPI, budget validation, user selection, human confirmation and itinerary generation. Confirmed travel plans receive a unique booking reference and are persisted in PostgreSQL using LangGraph checkpointing and a dedicated booking table.

The frontend and backend are packaged as separate Docker containers, stored in Amazon ECR and deployed as scalable Kubernetes services on Amazon EKS. Kubernetes Deployments, ClusterIP services, ConfigMaps, Secrets, NGINX Ingress and Horizontal Pod Autoscaling are used to manage application configuration, networking, availability and scaling.

GitHub is used for source-code management, while GitHub Actions automates the AWS authentication, Amazon ECR login, Kubernetes manifest rendering, EKS kubeconfig configuration, secret creation, deployment and rollout validation process

Services Used



MODULE 9: GENERATIVE AI ON AMAZON EKS

OVERVIEW

This module covers Kubernetes fundamentals and the deployment of containerized Generative AI applications on Amazon EKS. Students work with core resources, configuration, secrets, resource controls and Ingress.

1. INTRODUCTION

- Containers, orchestration and Kubernetes use cases
- Kubernetes and EKS for scalable GenAI services

2. KUBERNETES ARCHITECTURE

- Control plane, API server, scheduler and etcd
- Worker nodes, kubelet, runtime, pods and services

3. INSTALLATION

- Install kubectl and configure AWS access
- Create or access an EKS cluster and configure kubeconfig

4. WORKING WITH KUBERNETES

- Namespaces, pods, deployments, ReplicaSets and services
- Apply YAML, view logs, execute commands and perform rollouts

5. RESOURCE MANAGEMENT

- CPU and memory requests and limits
- Health probes, scaling and basic workload reliability

6. DAEMONSETS

- Run one workload instance on every selected node
- Logging, monitoring and node-agent use cases

7. CONFIGMAPS AND SECRETS

- Externalize application configuration
- Inject configuration and secrets as variables or mounted files

8. INGRESS EXAMPLE

- Expose multiple services through host-based or path-based routing
- Configure an Ingress controller and test routing

9. INGRESS

- Ingress resources, controllers and load balancers
- TLS, DNS, routing rules and production considerations

MODULE 10: Model Context Protocol (MCP)

OVERVIEW

This module introduces Model Context Protocol as a standard way for AI applications to connect with external tools, data sources and enterprise systems. Students learn the complete MCP architecture and build local and remote MCP servers through practical demonstrations.

1. WHY MODEL CONTEXT PROTOCOL?

- Enterprise information is distributed across GitHub, Jira, Slack, Google Drive, databases and cloud platforms
- Limitations of manually copying application code, tickets, schemas and documents into an LLM
- Security, outdated-context, scalability and action-execution problems
- Understanding the N clients × M tools integration-explosion problem
- How MCP provides a common interface between AI applications and external systems

2. EVOLUTION FROM LLMs TO MCP

- Traditional applications calling functions directly
- Using LLMs to generate code and responses
- Adding tools to create action-taking AI agents
- Limitations of building separate tool integrations for every AI client
- Using MCP as a reusable and standardized integration layer

3. MCP ARCHITECTURE

- MCP Host: the main AI application used by the user
- MCP Client: the connector inside the host that communicates with an MCP server
- MCP Server: the service that exposes tools, resources and prompts
- One host managing multiple MCP client-to-server connections
- End-to-end request flow from user request to tool execution and final response
- MCP architecture with GitHub, Jira, Slack, databases and cloud platforms

4. MCP BENEFITS AND LIMITATIONS

- Decoupling AI applications from external-system implementations
- Reusable integrations across multiple AI hosts
- Scalable and standardized client-server communication
- Improved discoverability of tools, resources and prompts
- Security, authentication, authorization and data-exposure considerations
- Dependency on server availability, tool quality and correct permission design

5. MCP PRIMITIVES

- Tools: actions the AI can invoke, such as creating issues or querying databases
- Resources: readable context such as files, schemas, documentation and configurations
- Prompts: reusable server-provided templates and recommended workflows
- Logging: structured server messages for debugging, monitoring and tracing
- Practical GitHub examples for tools, resources and prompts

6. MCP STANDARD OPERATIONS

- Discover available tools using tools/list
- Execute a selected tool using tools/call
- Discover readable resources using resources/list

- Read a selected resource using resources/read
- Discover prompt templates using prompts/list
- Retrieve a prompt template using prompts/get
- Request, response, notification and error-message behaviour

7. JSON-RPC 2.0 MESSAGE FORMAT

- Why MCP messages use structured JSON-RPC instead of natural-language communication
- Request identifiers, method names, parameters and results
- Matching response identifiers with request identifiers
- JSON-RPC notifications without response identifiers
- Standard error objects and invalid-parameter handling

8. MCP SESSION LIFECYCLE

- Initialization phase and client-server handshake
- Protocol-version negotiation
- Client and server capability negotiation
- Initialized notification and readiness for normal operations
- Operation phase for tools, resources and prompts
- Graceful shutdown for local and remote MCP servers
- Cancellation, error handling and session cleanup

9. MCP CAPABILITIES

- Roots for defining permitted folders, repositories and working locations
- Sampling for allowing the server to request LLM assistance through the client
- Elicitation for collecting missing information from the user
- Notifications for one-way server-to-client updates
- Sub-capabilities such as listChanged
- Using only capabilities negotiated during initialization

10. MCP TRANSPORT LAYER

- Difference between the MCP data layer and transport layer
- STDIO transport for local MCP servers
- Streamable HTTP for remote MCP servers
- Role of HTTP POST and optional streaming responses
- JSON-RPC as the common message format across transports
- Selecting transport based on deployment and sharing requirements

11. LOCAL MCP SERVERS

- Running the MCP host, client and server on the same machine
- Using STDIO for local process communication
- Accessing local files, SQLite databases, scripts, logs and project folders
- Connecting a local MCP server to Claude Desktop
- Testing local servers using MCP Inspector
- Local server startup, execution and shutdown

12. REMOTE MCP SERVERS

- Hosting an MCP server in a shared cloud environment
- Using Streamable HTTP for remote communication
- Designing reusable company-level HR, finance, travel and GitHub MCP servers
- Authentication, authorization and secure secret management
- Deploying and testing a remote FastMCP server
- Remote session and connection shutdown

13. FASTMCP VS MCP PYTHON SDK

- FastMCP for simplified syntax, decorators, rapid development and teaching
- MCP Python SDK for lower-level protocol control and production customization
- Installing FastMCP and the MCP SDK
- Selecting the framework based on project requirements

14. BUILDING MCP SERVERS WITH PYTHON

- Creating a Python virtual environment
- Initializing the project using uv
- Installing and verifying FastMCP

- Creating tools using decorators
- Exposing resources and reusable prompts
- Running the server locally
- Testing tools, resources and prompts using MCP Inspector
- Handling validation errors and structured responses

15. MCP CLIENT AND HOST INTEGRATION

- Understanding the host-client-server relationship
- Configuring Claude Desktop to connect with a local MCP server
- Manual MCP client configuration
- Tool discovery and tool selection by the LLM
- Reading resources and applying server-recommended prompts
- Viewing logs and tracing the execution flow

16. ENTERPRISE MCP USE CASES

- GitHub repository, issue and pull-request management
- Jira story retrieval and ticket updates
- Slack and Microsoft Teams notifications
- Google Drive and enterprise-document access
- MySQL and PostgreSQL schema or data access
- Terraform and Kubernetes operational tools
- HR, finance, travel and internal support assistants

LAB 1: SIMPLE MCP SERVER

Services and Technologies: Python, FastMCP, uv and MCP Inspector

Project Description: Build a simple MCP server that exposes roll-dice and addition tools. Students run the server locally, discover its tools through MCP Inspector and invoke the tools using structured MCP requests.

- Create and initialize the FastMCP project
- Implement roll_dice and add_numbers tools
- Run and test the MCP server locally
- Inspect tools and invoke operations using MCP Inspector

Project 7: MCP-Enabled AI Travel Planning and Booking Agent using LangGraph on Amazon EKS

Project Description

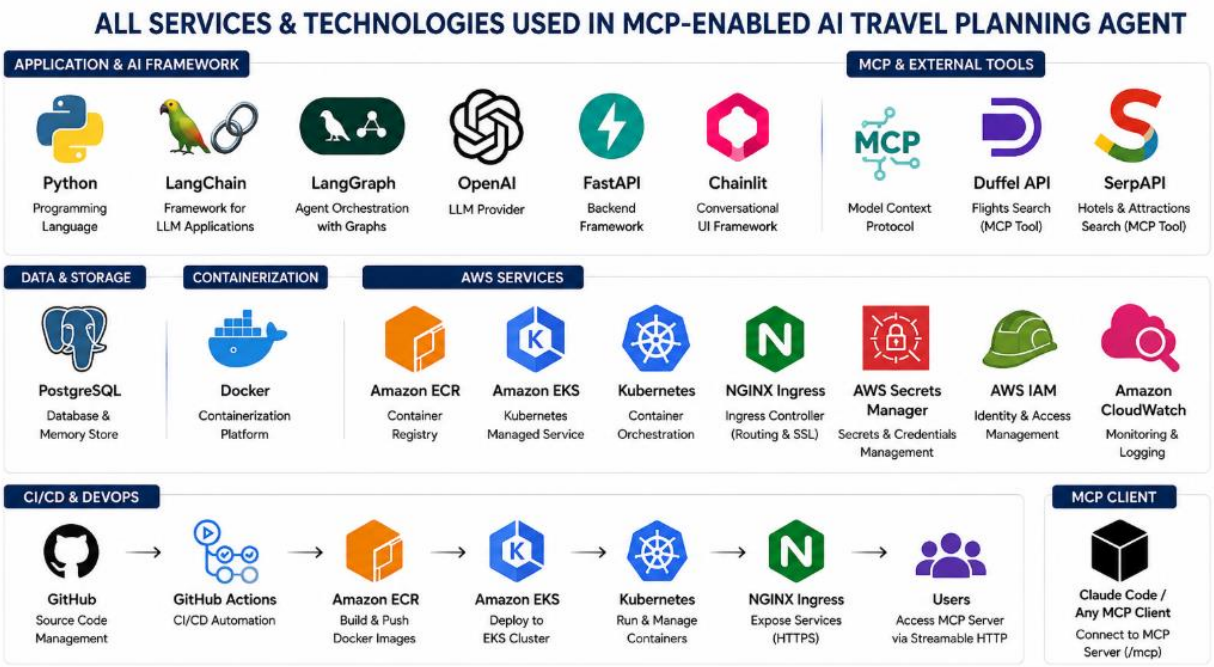
Develop an enterprise-grade Agentic AI travel assistant that enables users to plan and manage trips through natural-language requests from Claude Code or another MCP-compatible client. A Travel MCP Server exposes tools for starting a travel plan, searching and selecting flights and hotels, updating the available budget, retrieving sightseeing activities, confirming the booking and retrieving completed itineraries. The MCP server communicates with an internal FastAPI and LangGraph backend that manages airport-code identification, travel-date normalization, live flight search using Duffel API, hotel and activity search using SerpAPI, budget validation, itinerary generation and booking confirmation. Confirmed bookings receive a unique reference and are persisted in PostgreSQL using LangGraph checkpointing and a dedicated bookings table.

The LangGraph backend and Travel MCP Server are packaged as separate Docker containers, stored in Amazon ECR and deployed as independent Kubernetes services on Amazon EKS. The LangGraph service remains private inside the cluster, while the MCP server is exposed through NGINX Ingress using Streamable HTTP. Kubernetes Namespaces, Deployments, Services, ConfigMaps, Secrets and Horizontal Pod Autoscaling are used to manage application configuration, secure credentials, networking, availability and scalability.

GitHub is used for source-code management, while GitHub Actions provides the complete CI/CD workflow. The pipeline builds the LangGraph backend and MCP server Docker images, pushes them to Amazon ECR, updates the Kubernetes manifests, creates runtime secrets,

connects to the Amazon EKS cluster, deploys both services and validates the Kubernetes rollout. After deployment, Claude Code or another MCP-compatible client connects securely to the public /mcp endpoint to access the travel-planning tools.

Services Used



MODULE 11: AMAZON BEDROCK AND AWS BEDROCK AGENTCORE

OVERVIEW

This module teaches students how to build Generative AI applications with Amazon Bedrock and move a locally developed LangGraph agent into a secure, scalable enterprise runtime using AWS Bedrock AgentCore. The module combines concepts, Python demonstrations, cloud deployment and enterprise banking projects.

PART 1: AMAZON BEDROCK FOUNDATIONS

1. INTRODUCTION TO AMAZON BEDROCK

- Amazon Bedrock as a fully managed Generative AI service
- No model infrastructure or server management required

2. CORE FUNCTIONALITY

- Unified API for accessing models from different providers
- Amazon Nova and Amazon Titan model families
- Third-party foundation models available through Amazon Bedrock
- Text, image, video, speech and embedding models

3. AMAZON BEDROCK TECHNICAL CONCEPTS

- Foundation models and model modalities
- Tokens, token limits and context-window considerations
- Prompt Engineering and system instructions
- Inference parameters: temperature, top-p, top-k and maximum tokens
- Retrieval-Augmented Generation using Knowledge Bases

4. BUSINESS APPLICATIONS

- Enterprise knowledge-management systems
- Industry use cases in banking, healthcare, retail and technology

5. BEDROCK ARCHITECTURE AND INTEGRATIONS

- Application-to-Bedrock request and response flow

- Bedrock Runtime and Bedrock Agent Runtime
- Amazon S3 for document storage

PART 2: WORKING WITH AMAZON BEDROCK

6. MODEL ACCESS AND PLAYGROUND

- Enable and review foundation-model access
- Explore model capabilities in the Bedrock console
- Configure temperature, top-p and maximum-token values

7. LOCAL DEVELOPMENT SETUP

- Configure AWS CLI credentials and region
- Create a Python virtual environment
- Install Boto3, Botocore and python-dotenv
- Configure credentials securely in local systems or Google Colab
- Verify the active AWS identity using AWS STS
- Avoid hardcoding access keys and application secrets

8. CALLING FOUNDATION MODELS USING THE CONVERSE API

- Create a Bedrock Runtime client using Boto3
- Configure model ID, system prompt and user message
- Invoke a model using the Converse API
- Read assistant messages and stop reasons

9. MULTI-TURN CONVERSATION

- Maintain user and assistant message history
- Send complete conversation history with each request
- Use system prompts to control assistant behaviour
- Implement follow-up questions that depend on earlier responses
- Understand token-cost growth in multi-turn conversations
- Apply conversation-history trimming and summarization strategies

10. TOOL USE WITH AMAZON BEDROCK

- Create local Python functions as agent tools
- Define tool specifications and JSON input schemas
- Allow the model to select an appropriate tool
- Execute the tool in application code
- Return tool results to the model
- Generate a final natural-language response from tool output

11. BEDROCK KNOWLEDGE BASES

- Create an Amazon S3 bucket for enterprise documents
- Upload PDF, text and policy documents
- Create and configure a Bedrock Knowledge Base
- Configure a supported embedding model and vector store
- Synchronize the data source
- Retrieve relevant document chunks using Python
- Generate grounded answers with source references

12. BEDROCK GUARDRAILS

- Create Guardrails through the AWS console
- Configure denied topics, content filters and sensitive-information filters
- Apply Guardrails to user input and model output
- Combine Guardrails with Knowledge Base retrieval
- Test safe and restricted questions

13. BUILDING AN AGENT WITH STRANDS

- Configure a Bedrock model using the Strands Agents SDK
- Create reusable Python tools using decorators
- Build a Knowledge Base search tool
- Build an invoice-calculation verification tool
- Define agent instructions and tool-selection rules
- Generate an auditable invoice-review response

PART 3: AWS BEDROCK AGENTCORE

14. WHY AWS BEDROCK AGENTCORE?

- Difference between demonstration agents and production agents
- Challenges of serving thousands of users
- Secure session isolation and multi-user access
- Safe access to internal APIs and enterprise tools
- Managed user and session memory
- Auditability, tracing, governance and cost visibility
- Production deployment without rewriting the LangGraph agent

15. AGENTCORE CAPABILITIES

- AgentCore Runtime for secure and scalable agent hosting
- AgentCore Gateway for controlled access to enterprise tools
- AgentCore Memory for managed short-term and long-term context
- AgentCore Identity for authentication and authorization
- AgentCore Observability for tracing and troubleshooting
- Framework-agnostic and model-agnostic architecture
- Using LangGraph with OpenAI while hosting the agent on AWS

16. ENTERPRISE BANKING AGENT USING LANGGRAPH AND OPENAI

- Create a LangGraph-powered banking operations agent
- Use OpenAI as the reasoning model
- Create tools for customer profiles, transactions, policies and service cases
- Allow the agent to select tools dynamically
- Differentiate safe, medium-risk and high-risk operations
- Return final responses with audit information
- Store API keys using environment variables

17. AGENTCORE RUNTIME - LOCAL DEVELOPMENT

- Understand BedrockAgentCoreApp and the runtime entry point
- Wrap the existing LangGraph agent without changing core business logic
- Accept prompt and customer ID through the invocation endpoint
- Create the AgentCore project structure
- Configure requirements, environment variables and Dockerfile
- Run the project using agentcore dev
- Test the local /invocations endpoint using curl

18. DEPLOYING AGENTCORE RUNTIME TO AWS

- Create an Amazon ECR repository
- Build and push the AgentCore Runtime container image
- Create the AgentCore Runtime execution role
- Configure ECR pull and CloudWatch permissions
- Create the runtime using the container image
- Capture and manage the runtime ARN
- Invoke the runtime using AWS CLI and Boto3
- Pass a valid runtime session ID for session isolation

19. RUNTIME SESSIONS, LOGS AND TROUBLESHOOTING

- Understand AgentCore Runtime sessions and isolated execution
- Trace requests from the client to the runtime entry point
- Understand container invocation through /invocations
- Inspect CloudWatch log groups and runtime logs
- Troubleshoot missing environment variables and model credentials
- Troubleshoot container startup and application errors
- Validate final responses and audit details

20. INTRODUCTION TO AGENTCORE GATEWAY

- Problem of duplicating tools across multiple agents
- Separating enterprise tools from agent application code
- AgentCore Gateway as a secure tool bridge
- Expose Lambda functions, REST APIs, MCP servers and HTTP services

- Convert OpenAPI operations into callable agent tools
- Control which tools and backend systems an agent can access

21. ENTERPRISE BANKING TOOL SERVICE

- Create a FastAPI service for banking operations
- Build customer-profile and transaction-search endpoints
- Build policy-search and service-case endpoints
- Store structured banking data in PostgreSQL or Neon
- Store policy documents in an Amazon Bedrock Knowledge Base
- Use Pydantic for request validation
- Package the FastAPI application as an AWS Lambda container image

22. FASTAPI, LAMBDA AND API GATEWAY DEPLOYMENT

- Create the Lambda-compatible FastAPI application using Mangum
- Build and push the container image to Amazon ECR
- Create the Lambda execution role and Bedrock retrieval permissions
- Create an AWS Lambda function from the ECR image
- Expose Lambda through Amazon API Gateway

23. CONFIGURING AGENTCORE GATEWAY

- Create an AgentCore Gateway
- Configure Gateway credentials and authentication
- Add the banking OpenAPI specification as a Gateway target
- Map API operations to agent tools
- Discover tools using tools/list
- Invoke Gateway tools using tools/call
- Validate PostgreSQL and Knowledge Base-backed operations

Project 8: AgentCore Gateway-Enabled Enterprise Banking Assistant

Project Description

Develop an enterprise-grade Agentic AI banking assistant using Python, LangGraph and OpenAI, deployed securely on AWS Bedrock AgentCore Runtime. The assistant helps customers retrieve account details, search transactions, understand banking policies, check loan information and create service requests through natural-language conversations.

Structured customer and transaction data is stored in PostgreSQL, while banking policies and documents are retrieved through Amazon Bedrock Knowledge Bases. FastAPI exposes approved banking operations through AWS Lambda and Amazon API Gateway, and AgentCore Gateway converts these APIs into centrally managed tools that the agent can securely discover and invoke.The application is containerized using Docker, stored in Amazon ECR and monitored through Amazon CloudWatch. AWS IAM controls service permissions, while AgentCore Runtime provides scalable execution, session isolation and secure agent deployment.

Services Used