UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Chalice Framework: Serverless Applications in Python

Gabriel Guerra, Alexis Vendrix, Joan Rodríguez, Nashly González

Content

1

What is Chalice?

Get to know the framework Chalice

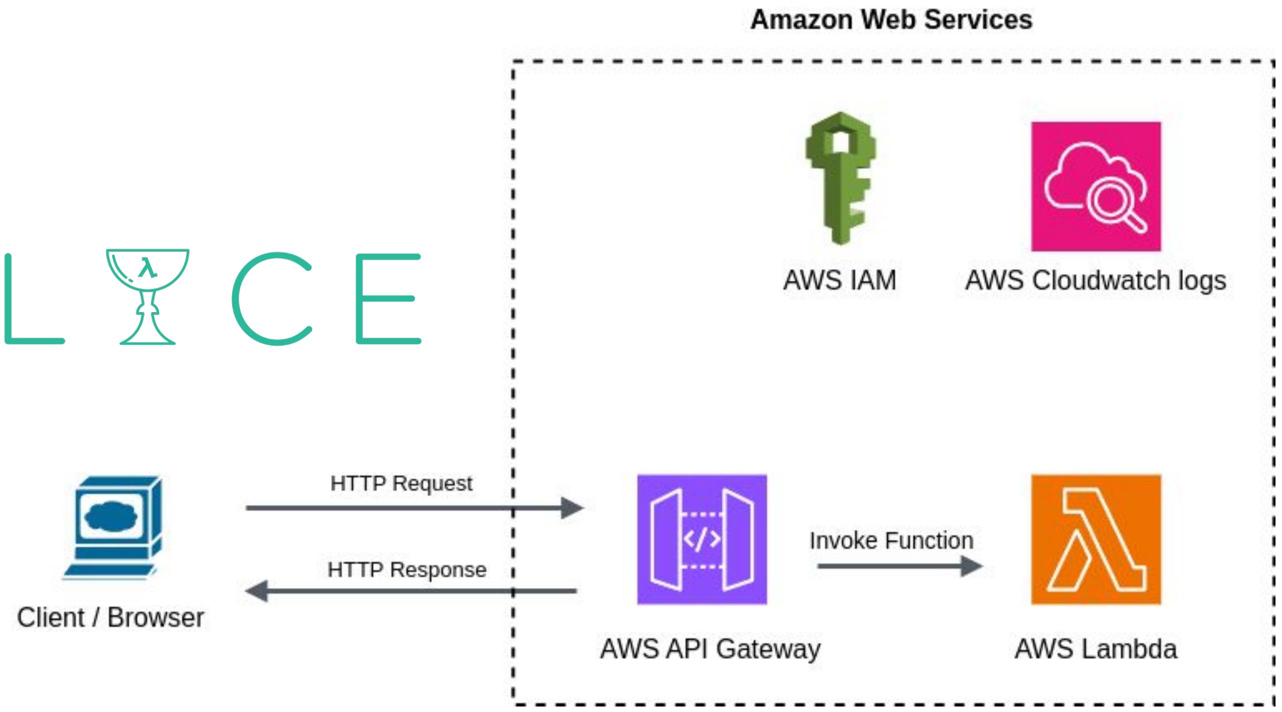
What is Chalice and why does it matter?

- Helps developers create and deploy serverless applications quickly.
- Micro Framework developed by Amazon.
- Integrates with various AWS Services.
- Programmed with Python.

CHALICE

How does it work?

CHALICE

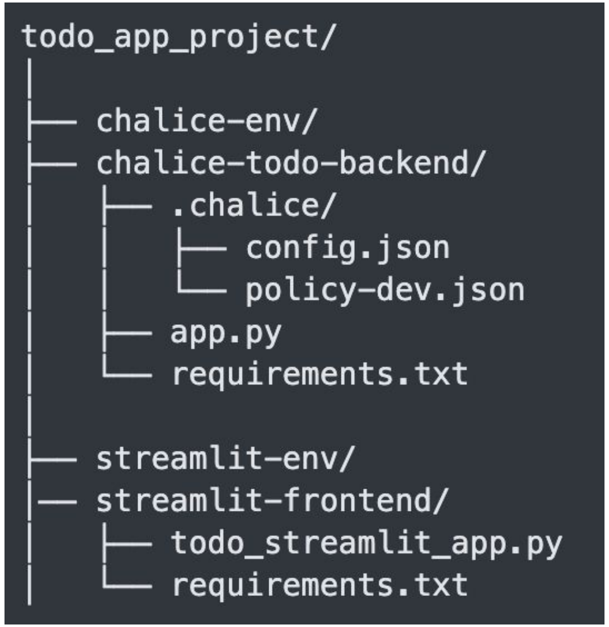


2

Tutorial: Deploying a Chalice App

Step-by-step: from setup to deployment

Getting Ready: Tools and Structure



```
todo_app_project/
│
├── chalice-env/
├── chalice-todo-backend/
│   ├── .chalice/
│   │   ├── config.json
│   │   └── policy-dev.json
│   ├── app.py
│   └── requirements.txt
│
├── streamlit-env/
│── streamlit-frontend/
│   ├── todo_streamlit_app.py
│   └── requirements.txt
```

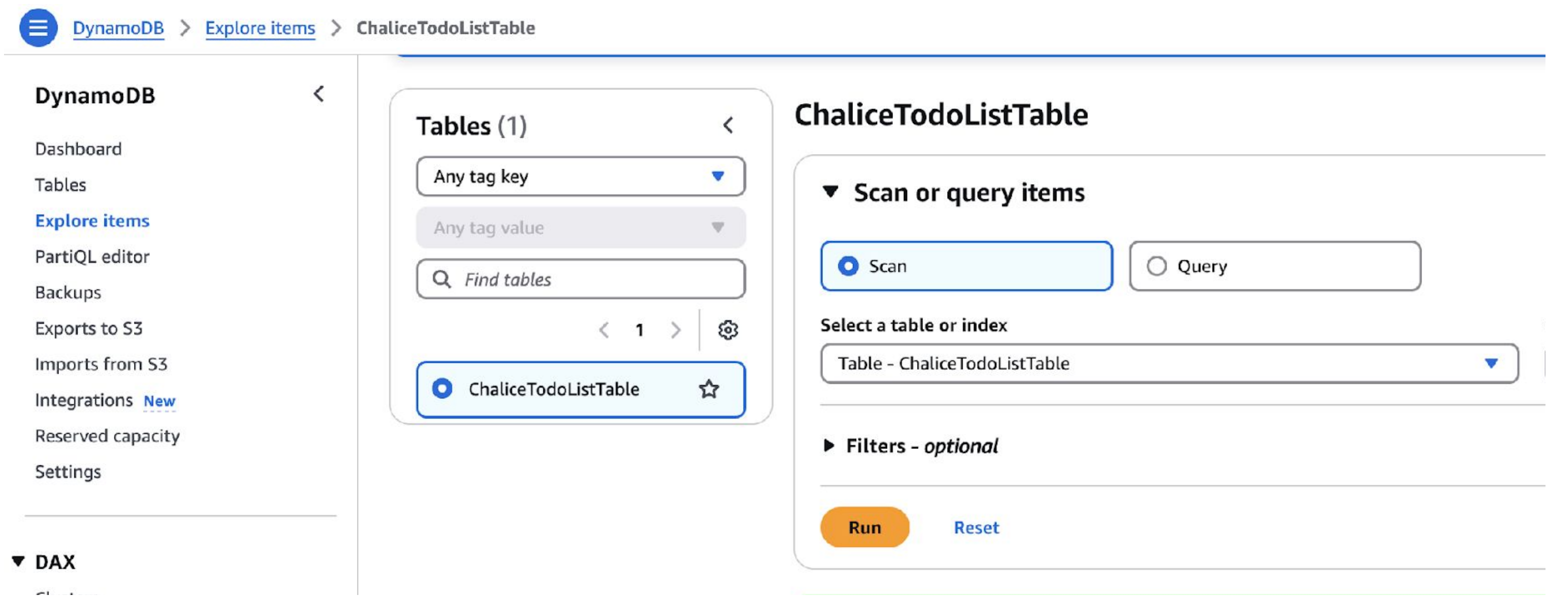
CHALICE

DynamoDB

amazon web services™

Streamlit

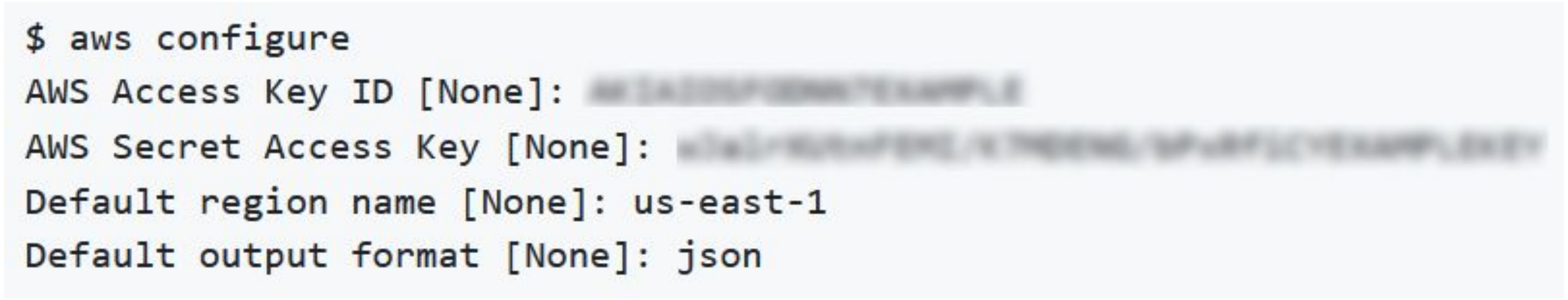
Step 1: Create and Configure DynamoDB Table



Select a table or index: Table - ChaliceTodoListTable

Scan [✓]

Step 2: Set Up AWS Credentials for Development



```
$ aws configure
AWS Access Key ID [None]: [redacted]
AWS Secret Access Key [None]: [redacted]
Default region name [None]: us-east-1
Default output format [None]: json
```

Step 3: Create the Chalice Backend

Writing the Application Code
(app.py)

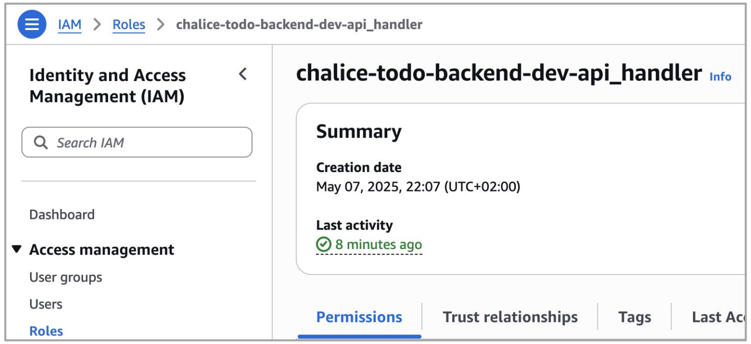
Define Dependencies
(requirements.txt for backend)

Configure IAM Permissions for Chalice
(config.json and policy-dev.json)

Deploy the Chalice Backend

Chalice deploy

1



IAM > Roles > chalice-todo-backend-dev-api_handler

Identity and Access Management (IAM)

Search IAM

Dashboard

Access management

User groups

Users

Roles

chalice-todo-backend-dev-api_handler Info

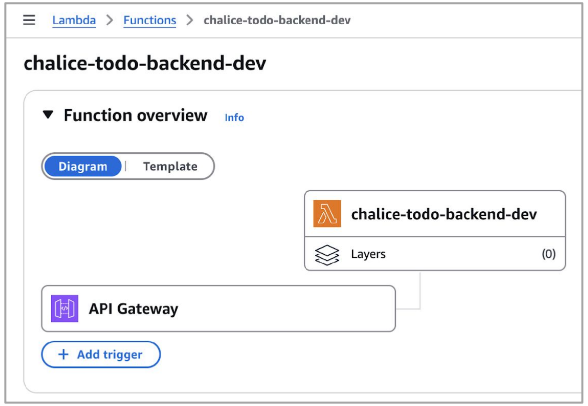
Summary

Creation date
May 07, 2025, 22:07 (UTC+02:00)

Last activity
8 minutes ago

Permissions | Trust relationships | Tags | Last Ac

2



Lambda > Functions > chalice-todo-backend-dev

chalice-todo-backend-dev

Function overview Info

Diagram | Template

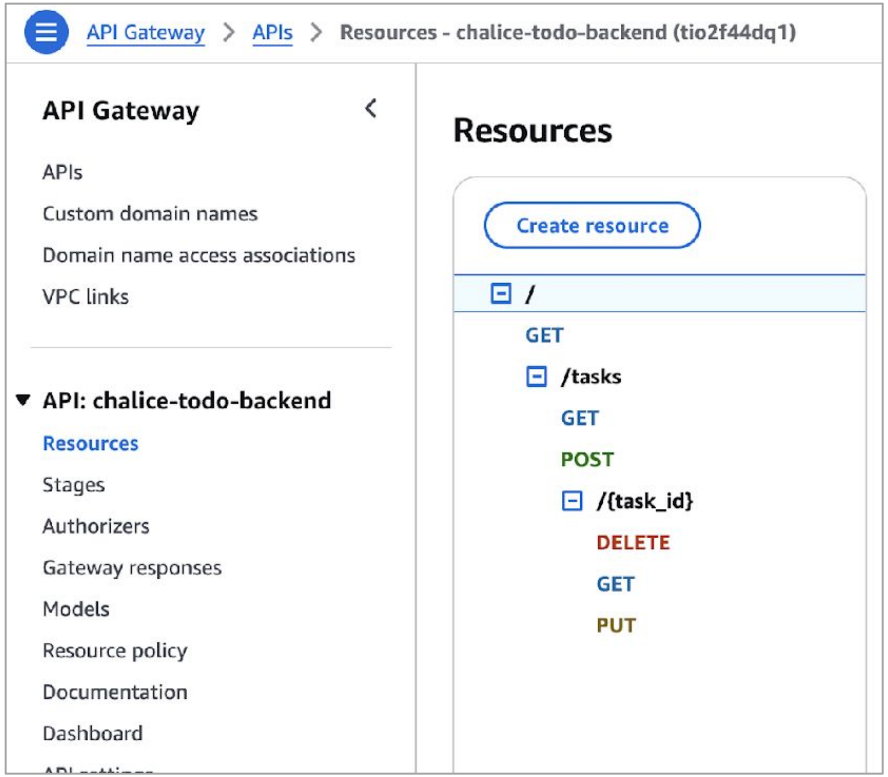
chalice-todo-backend-dev

Layers (0)

API Gateway

+ Add trigger

3



API Gateway > APIs > Resources - chalice-todo-backend (tio2f44dq1)

API Gateway

APIs

Custom domain names

Domain name access associations

VPC links

API: chalice-todo-backend

Resources

Stages

Authorizers

Gateway responses

Models

Resource policy

Documentation

Dashboard

Resources

Create resource

```
/
  GET
  /tasks
    GET
    POST
    /{task_id}
      DELETE
      GET
      PUT
```

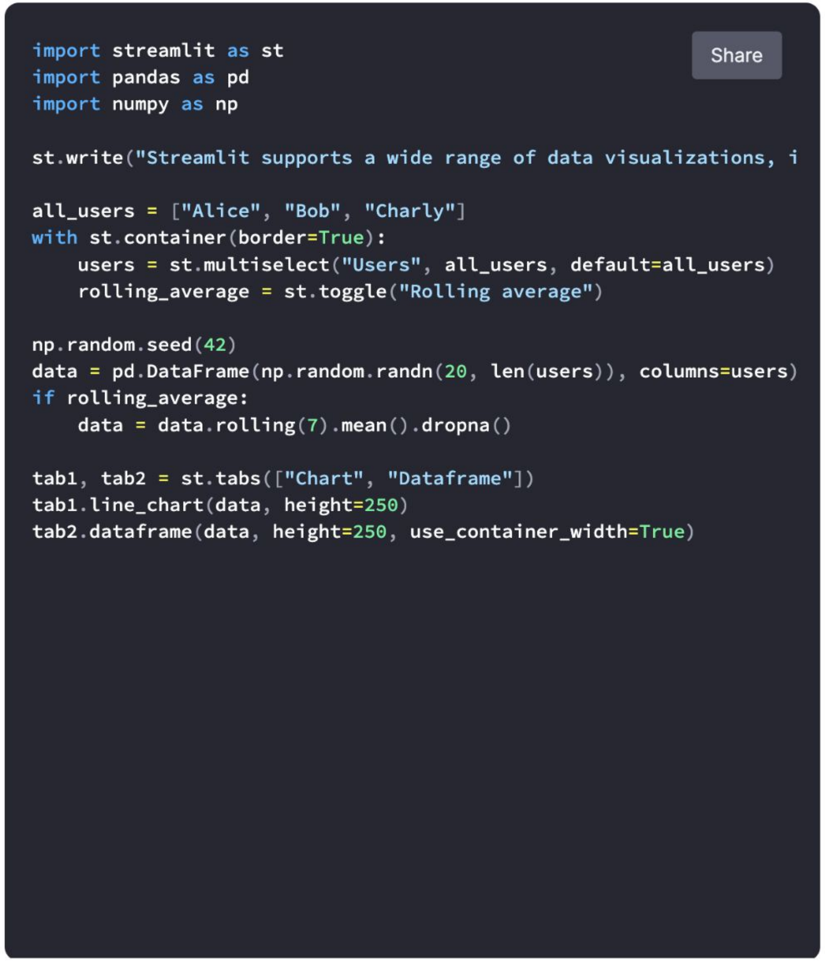
Step 4: Create frontend with Streamlit

Set Up the Streamlit Project

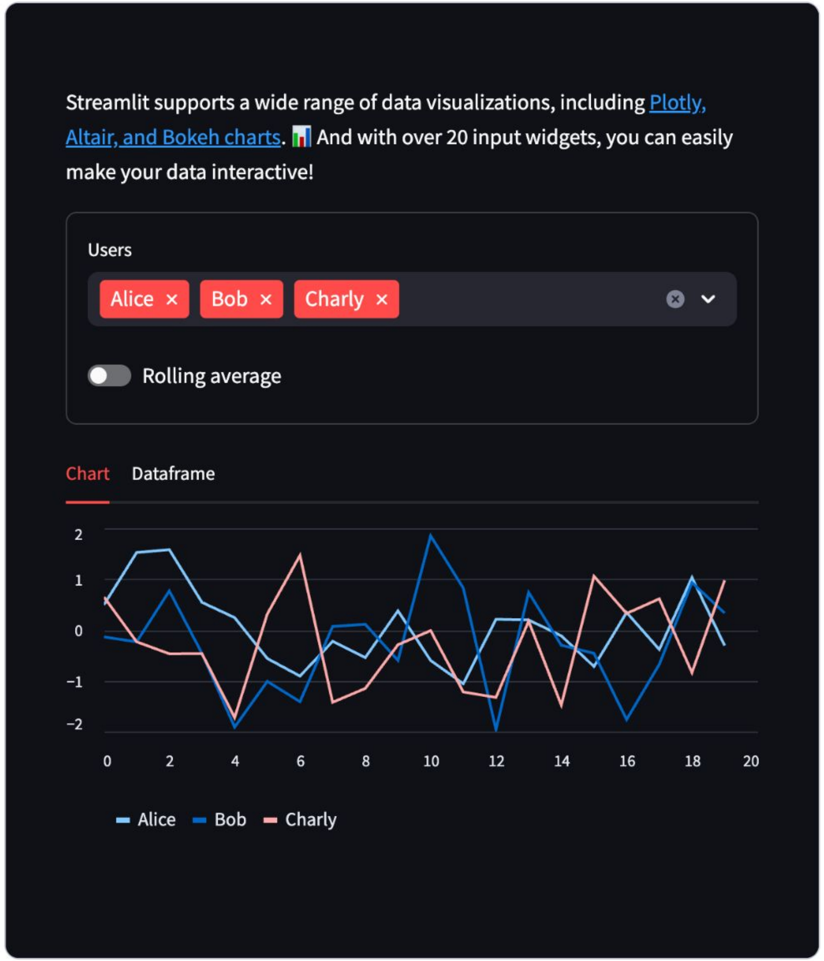
Write the Streamlit application code (todo_streamlit_app.py)

Define Dependencies (requirements.txt for frontend)

Run Locally and Deploy Streamlit (for test purpose first)



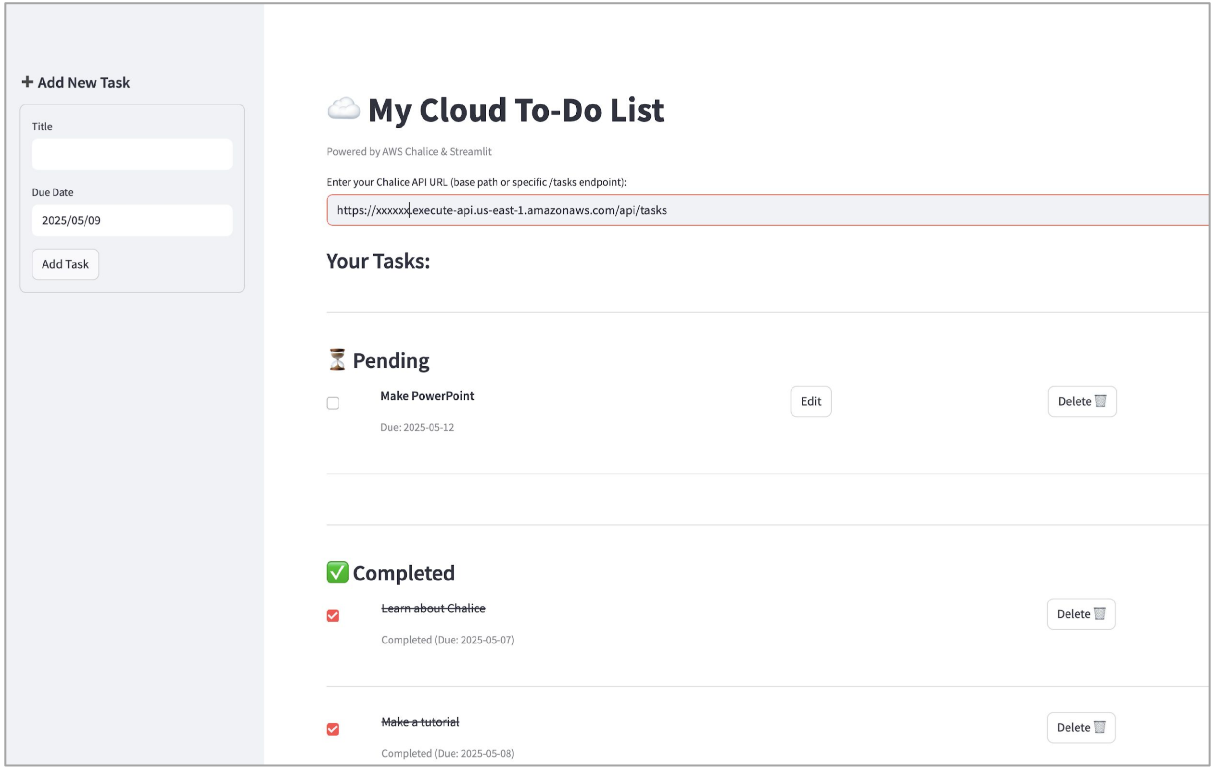
```python
import streamlit as st
import pandas as pd
import numpy as np

st.write("Streamlit supports a wide range of data visualizations, i

all_users = ["Alice", "Bob", "Charly"]
with st.container(border=True):
    users = st.multiselect("Users", all_users, default=all_users)
    rolling_average = st.toggle("Rolling average")

np.random.seed(42)
data = pd.DataFrame(np.random.randn(20, len(users)), columns=users)
if rolling_average:
    data = data.rolling(7).mean().dropna()

tab1, tab2 = st.tabs(["Chart", "Dataframe"])
tab1.line_chart(data, height=250)
tab2.dataframe(data, height=250, use_container_width=True)
```



Streamlit supports a wide range of data visualizations, including Plotly, Altair, and Bokeh charts. 📊 And with over 20 input widgets, you can easily make your data interactive!

Step 5: Deploying the Streamlit frontend

To visualize and test the application, you can go on your Streamlit account and find your application listed.


+ Add New Task

Title

Due Date

2025/05/09

Add Task

☁️ My Cloud To-Do List

Powered by AWS Chalice & Streamlit

Enter your Chalice API URL (base path or specific /tasks endpoint):

https://xxxxxx.execute-api.us-east-1.amazonaws.com/api/tasks

Your Tasks:

⏳ Pending

☐ Make PowerPoint Edit Delete 🗑️

Due: 2025-05-12

✅ Completed

☑ ~~Learn about Chalice~~ Delete 🗑️

Completed (Due: 2025-05-07)

☑ ~~Make a tutorial~~ Delete 🗑️

Completed (Due: 2025-05-08)

3

Pros, Limitations, and Use Cases

What works well and when to use it

Pros and Limitations

- Simple and fast to set up with Python.
- Native integration with AWS services (Lambda, API Gateway, etc.).
- Ideal for building small REST APIs and microservices.

- AWS-only — not cross-platform.
- Not ideal for large or complex applications.
- Limited customizability compared to full frameworks like FastAPI.

Use Cases

Task management apps or CRUD backends
(like this project)

Backend APIs for Streamlit or mobile/web apps

Event-driven architectures, webhooks, automation tasks

Rapid prototyping and testing of serverless concepts

4

Learning Resources

Where to learn more and keep building

Learning Resources

AWS Chalice Documentation

Full reference for commands, deployment, and configuration examples.

Streamlit Documentation

Guide for building interactive UIs in pure Python.

DynamoDB Documentation

Reference for building and managing NoSQL key-value databases on AWS.

5

Conclusions

Key takeaways from our experience

Conclusions

- We experienced full serverless application development using AWS Chalice, Lambda, API Gateway, and DynamoDB — without managing any servers.
- We built a complete frontend in Streamlit, entirely in Python, demonstrating API integration, UI interaction, and fast prototyping.
- We practiced real-world cloud principles: API-first, microservices, cloud deployment, and IaC with Chalice.

THANKS!

Any questions?