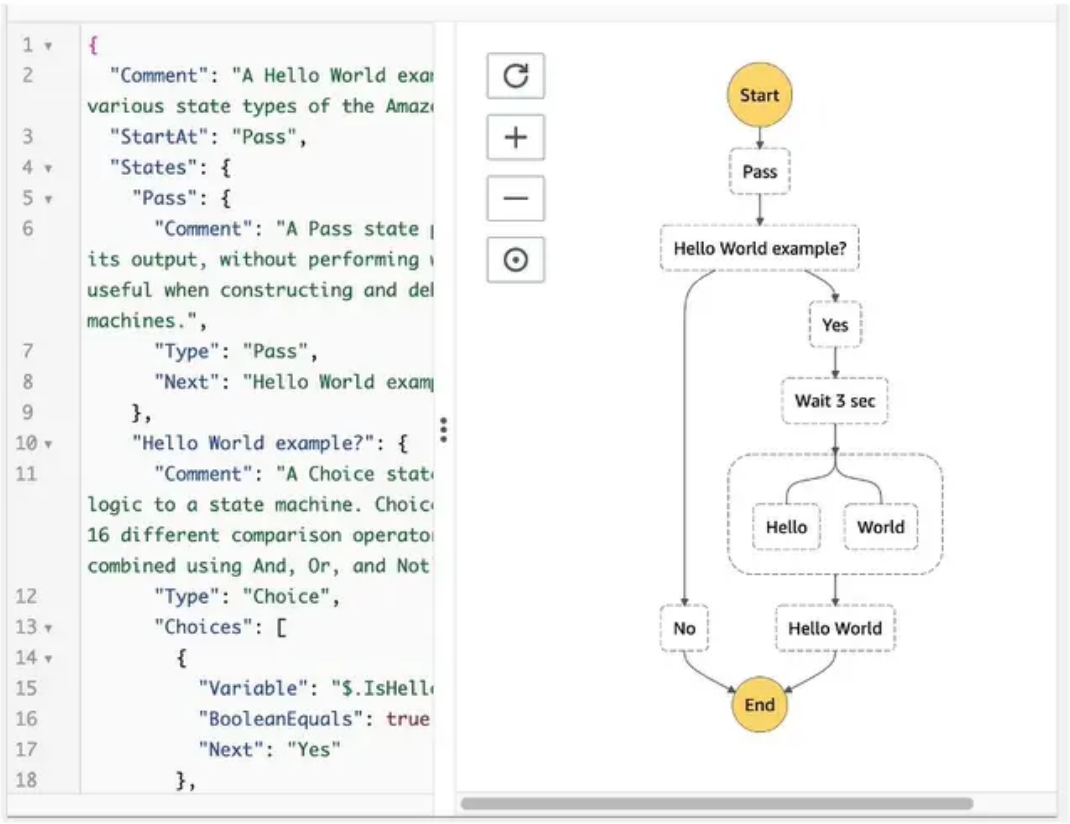
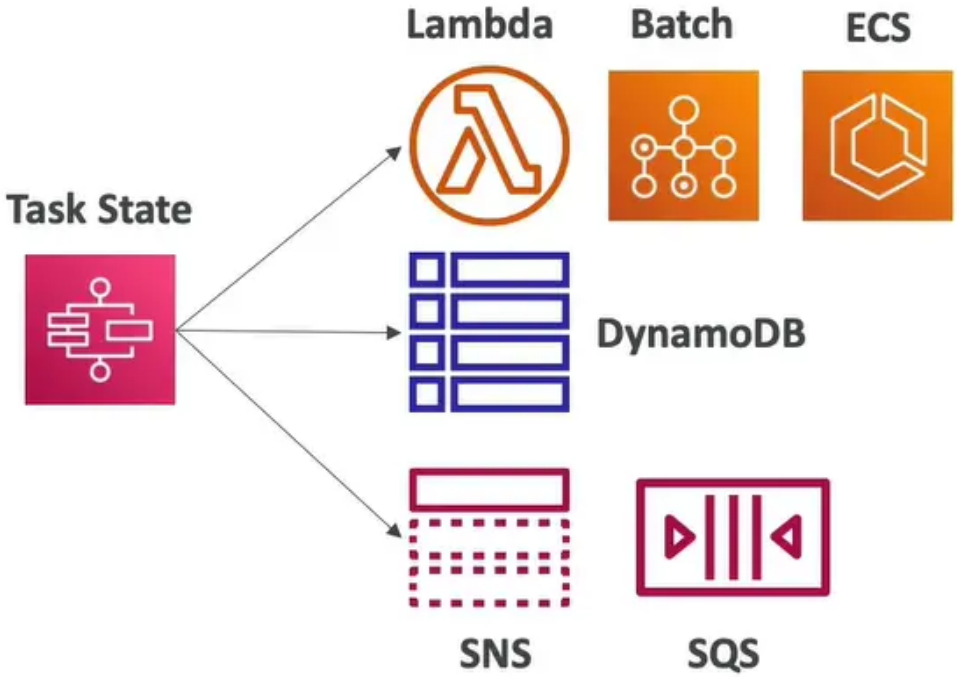
Other Serverless:Step Function & AppSync

AWS Step Function



- model your workflowss as state machines (one per workflow)
  - order fulfillment, data processing
  - web applications, any workflow
- written in JSON
- visualization of the workflow and the execution of the workflow,as well as history
- start workflow with SDK call, API Gateway, Event Bridge (CloudWatch Event)

Task States



- do some work in your state machine
- invoke one AWS service
  - can invoke a Lambda function
  - run an AWS Batch job
  - run an ECS task and wait for it to complete
  - insert an item from DynamoDB
  - publish message to SNS, SQS
  - launch another step function workflow...
- run an one Activity
  - EC2 , Amazon ECS, on–premises
  - Activities poll the Step function for work
  - Activities send result back to Step Functions

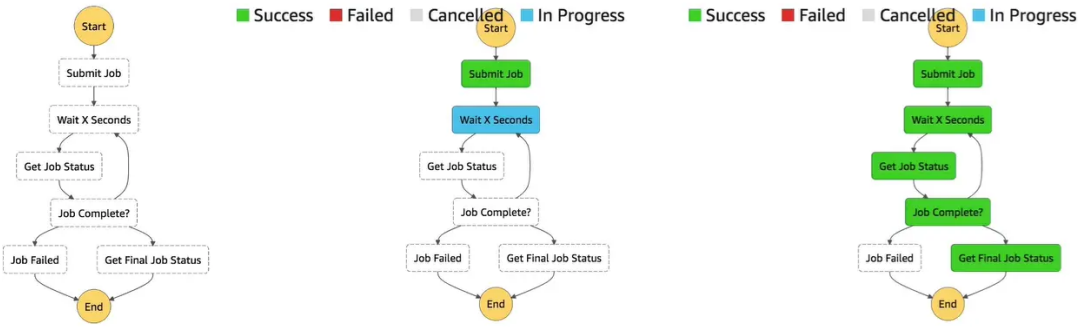
Example – Invoke Lambda Function

```
"Invoke Lambda function": {
    "Type": "Task",
    "Resource": "arn:aws:states:::lambda:invoke",
    "Parameters": {
        "FunctionName":
        "arn:aws:lambda:REGION:ACCOUNT_ID:function:FUNCTION_NAME",
        "Payload": {
            "Input.$": "$"
        }
    },
    "Next": "NEXT_STATE",
    "TimeoutSeconds": 300
}
```

States

- Choice State – Test for a condition to send to a branch (or default branch)
- Fail or Succeed State – Stop execution with failure or success
- Pass State – simply pass its input to its output or inject [注入] some fixed data,without performing work
- Wait State – provide a delay for a certain amount of time or until a specified time / date
- Map state – Dynamically [动态] iterate [迭代] steps.
- parallel State – begin parallel branches of execution

visual workflow in step function



Error Handing in step function

- any state can encounter runtime errors for various reasons
  - state machine definition issues (for example, no matching rule in a choice state)
  - task failures (for example, an exception in a Lambda function)
  - transient [短暂的] issues (for example,network partition events)
- use Retry (to retry failed state) and Catch (tansition to failure path) in the State Machine to handle the errors instead of inside the Application Code
- predefined error codes
  - State.ALL : matches any error name
  - States.Timeout: Task ran longer than TimeoutSeconds or no heartbeat received
  - States.TaskFailed: execution failure
  - State.Permissions : insufficient privileges to execute code
- the state may report is own errors

Retry (Task or Parallel State)

```
"HelloWorld": {
  "Type": "Task",
  "Resource": "arn:aws:lambda:REGION:ACCOUNT_ID:function:FUNCTION_NAME",
  "Retry": [
    {
      "ErrorEquals": ["CustomError"],
      "IntervalSeconds": 1,
      "MaxAttempts": 2,
      "BackoffRate": 2.0
    },
    {
      "ErrorEquals": ["States.TaskFailed"],
      "IntervalSeconds": 30,
      "MaxAttempts": 2,
      "BackoffRate": 2.0
    },
    {
      "ErrorEquals": ["States.ALL"],
      "IntervalSeconds": 5,
      "MaxAttempts": 5,
      "BackoffRate": 2.0
    }
  ],
  "End": true
}
```

- evaluated [评估] from top to bottom
- ErrorEquals : match a specific kind of error
- IntervalSeconds : inital delay before retrying
- BackoffRate : multiple the delay after each retry
- MaxAttempts : default to 3,set to 0 for never retried
- when max attempts are reached, the Catch kicks in[开始]

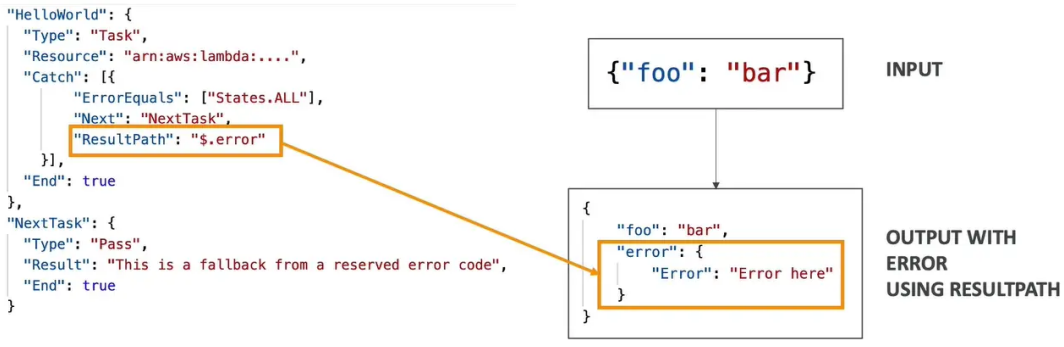
Catch （Task or Parallel State)

```
"HelloWorld": {
  "Type": "Task",
  "Resource": "arn:aws:lambda:....",
  "Catch": [
    {
      "ErrorEquals": ["CustomError"],
      "Next": "CustomErrorFallback"
    },
    {
      "ErrorEquals": ["States.TaskFailed"],
      "Next": "ReservedTypeFallback"
    },
    {
       "ErrorEquals": ["States.ALL"],
       "Next": "NextTask",
       "ResultPath": "$.error"
    }
  ],
  "End": true
},
"CustomErrorFallback": {
  "Type": "Pass",
  "Result": "This is a fallback from a custom lambda function exception"
  "End": true
},
```

- evaluated from top to bottom
- ErrorEquals : match a specific kind of error
- Next : state to send to
- ResultPath – A path that determines what input is sent to the state specified in the next field

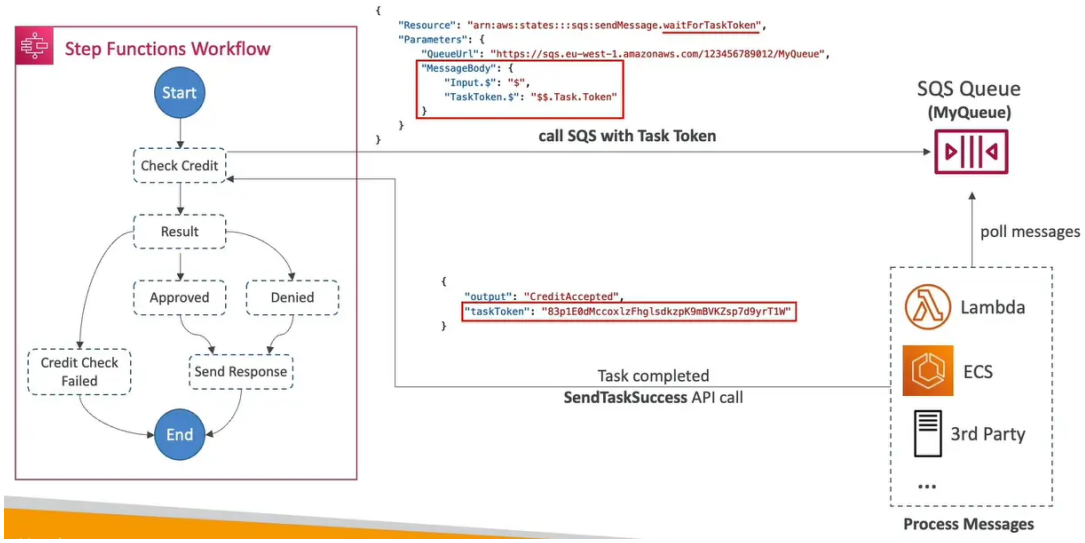
ResultPath

- include the error in the input

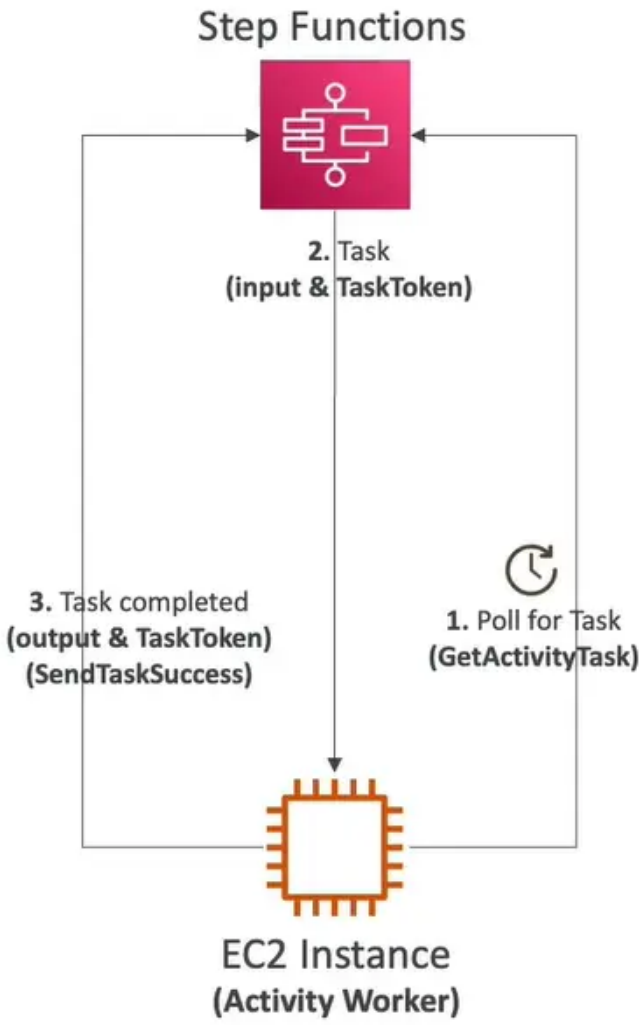


Wait for Task Token

- allows you to pause Step Functions during a Task until a Task Token is returned
- task might wait for other AWS service, human approval, 3rd party integration, call legacy systems ...
- append [附加] .waitForTaskToken to the Resource field to tell Step Functions to wait for the Task Token to be returned
- `"Resource": "arn:aws:states:::sqs:sendMessage.waitForTaskToken"`
- task will pause until it receives that Task Token back with a SendTaskSuccess or SendTaskFailure API call



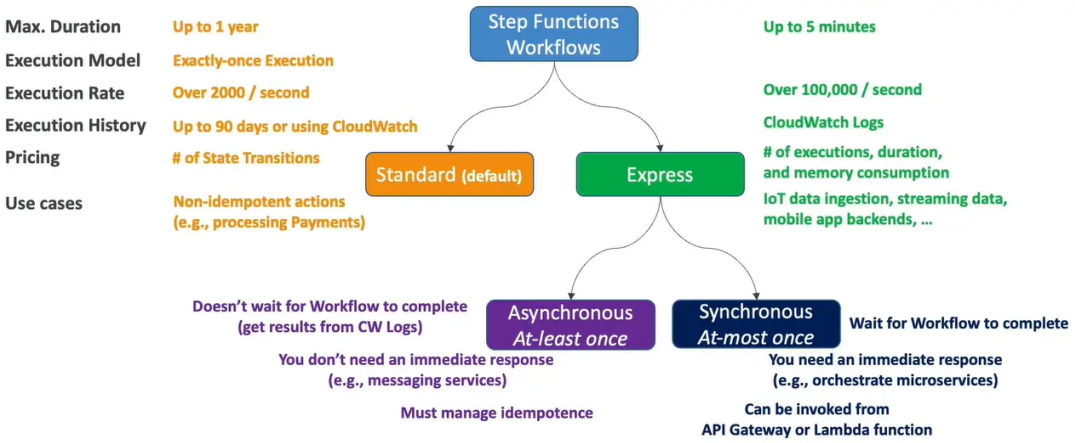
Activity Tasks



- enables you to have the task work performed by an *Activity Worker*
- *Activity Worker* apps can be running on EC2, Lambda, mobile device..
- activity worker poll for a Task using GetActivityTask API
- After Activity Worker completes its work, it sends a response of its success / failure using SendTaskSuccess or SendTaskFailure
- to keep the task active:
  - configure how long a task can wait by setting TimeoutSeconds

- periodically send a heartbeat from your Activity Worker using SendTaskHeartBeat within the time you sent in HeartBeatSeconds
- by configuring a long TimeoutSeconds and actively sending a hearbeat, Activity Task can wait up to 1year

Standard vs Express [特快]

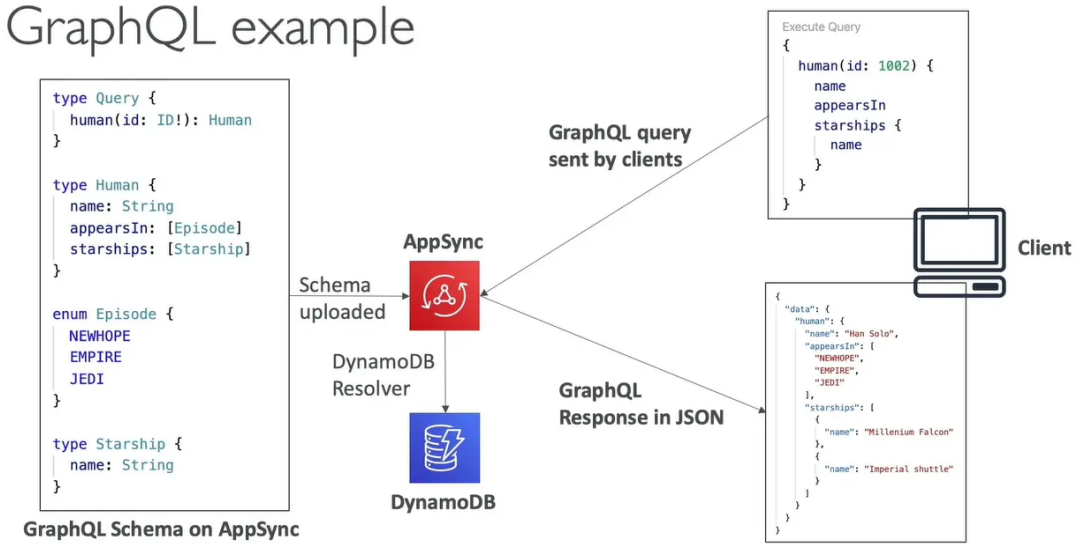


AWS APPSync

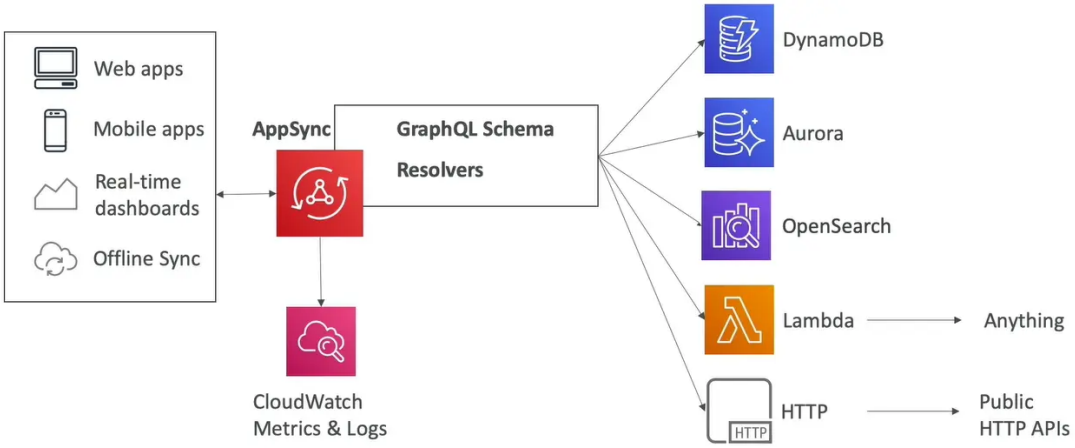
Overview

- AppSync is a managed service that uses GraphQL
- GraphQL makes it easy for applications to get exactly the data they need
- this includes combining data from one or more sources
  - NoSQL data stores, Relational databases, HTTP APIs..
  - integrates with DynamoDB, Aurora, OpenSearch & others
  - custom sources with AWS Lambda
- Retrieve data in real-time with WebSocket or MQTT on WebSocket
- for mobile apps: local data access & data synchronization
- it all stats with uploading one GraphQL schema

example



Diagram

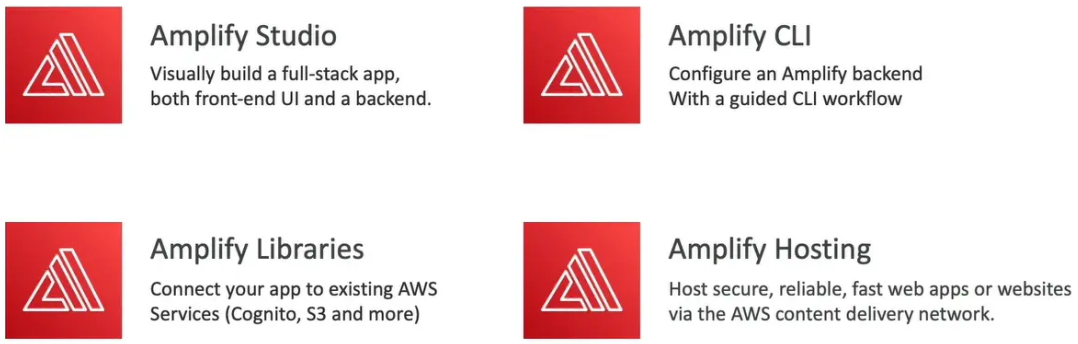


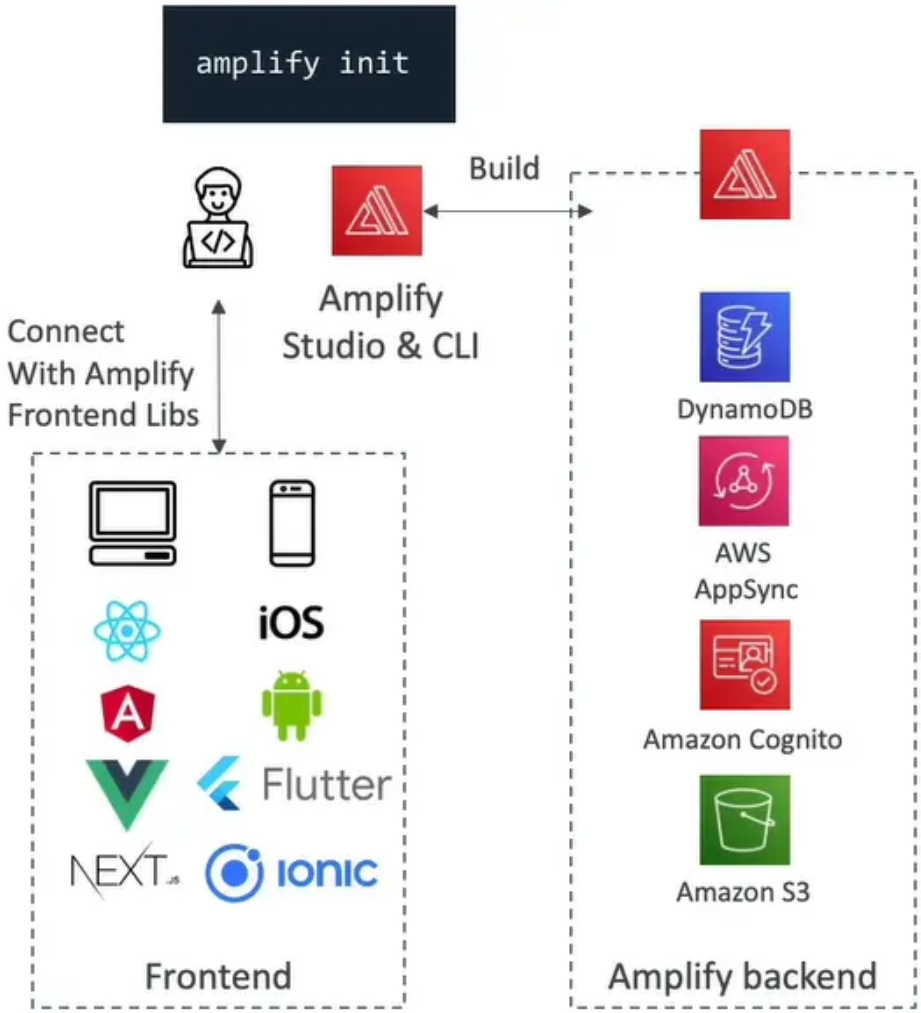
Security

- there are four ways you can authorize applications to interact with your AWS AppSync GraphQL API:
- API_KEY
- AWS_IAM : IAM users / roles / cross-account access
- OPEN_CONNECT : OpenID Connect provider / JSON Web Token
- AMAZON_COGNITO_USER_POOLS

- for custom domain & HTTPS, use CloudFront in front of AppSync

AWS Amplify – Create mobile and web applications





- set of tools to get started with creating **mobile and web application**
- "Elastic Beanstalk for mobile and web applications"
- must-have features such as **data storage, authentication, storage, and machine-learning,** all powered by AWS services
- **front-end libraries** with ready-to-use components for React.js, Vue,Javascript,IOS,Android,Flutter,etc..
- incorporates AWS best practices to for reliability, security,scalability
- build and deploy with the **Amplify CLI** or **Amplify Studio**

important features

Authentication

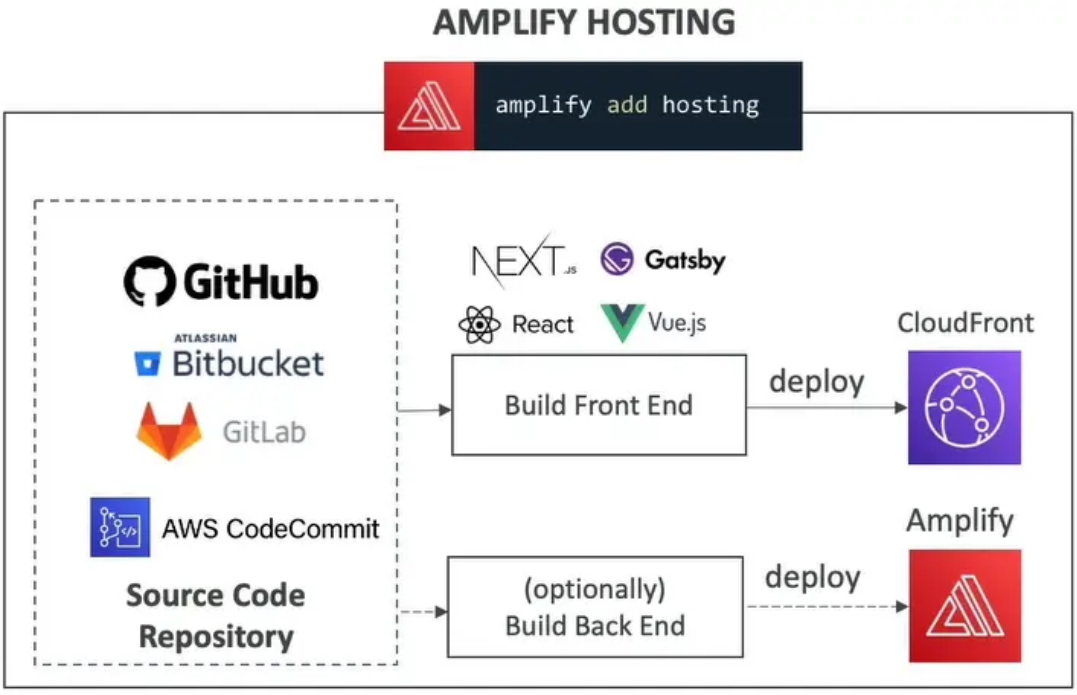
```
amplify add auth
```

- leverages Amazon Cognito
- user registration, authentication, account recovery & other operations
- support MFA, Social Sign-in, etc...
- Pre-built UI components
- Fine-grained authorization

DataStore

- leverages Amazon AppSync and Amazon DynamoDB
- work with local data and have **automatic synchronization to the cloud** without complex code
- powered by GraphQL
- offline and real-time capabilities
- visual data modeling w/ Amplify Studio

AWS Amplify Hosting

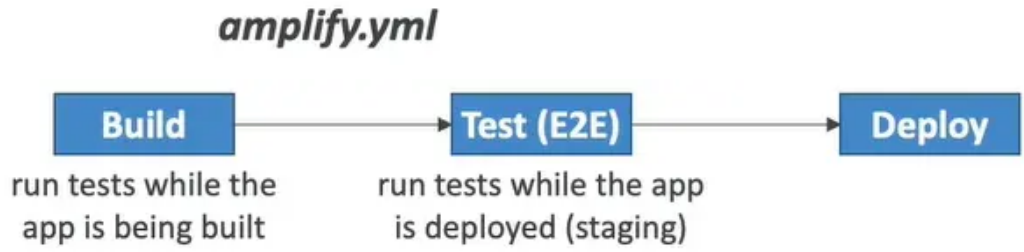


- build and host modern web apps
- CICD (build, test, deploy)
- pull request previews
- custom domains
- monitoring
- redirect and custom headers
- password protection

End-to-End (E2E) Testing

```yaml
test:
  phases:
    preTest:
      commands:
        - npm ci
        - npm install -g pm2
        - npm install -g wait-on
        - npm install mocha mochawesome ...
        - pm2 start npm -- start
        - wait-on http://localhost:3000
    test:
      commands:
        - 'npx cypress run --reporter ...'
    postTest:
      commands:
        - npx mochawesome-merge cypress/...
        - pm2 kill

  artifacts:
    baseDirectory: cypress
    configFilePath: "**/mochawesome.json"
    files:
      - "**/*.png"
      - "**/*.mp4"
```



- run end-to-end (E2E) tests in the test phase in Amplify [放大]
- catch regressions [回归] before pushing code to production
- use the test step to run any test commands at build time (amplify.yml)
- integrated with Cypress testing framework
  - allows you to generate UI report for your tests