E6156 – Topics in SW Engineering: Cloud Computing

Lecture 8: CloudFront, Certificates, DNS, Service Composition, Step Functions

© Donald F. Ferguson, 2021

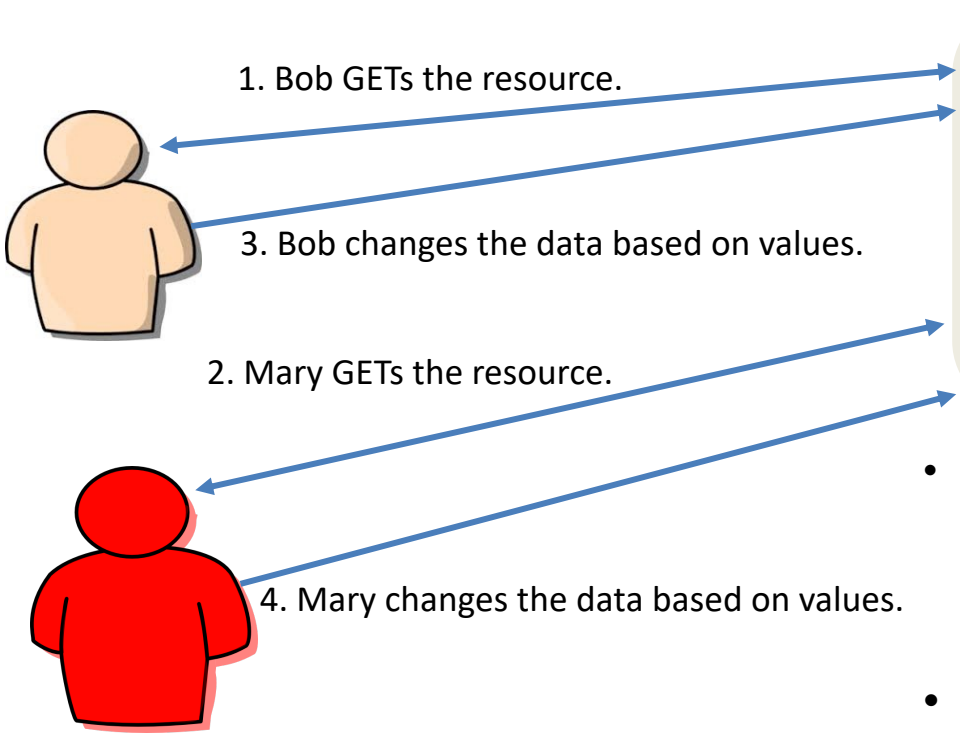
Will will start in a minute.

© Donald F. Ferguson, 2021

ETag, Conditional Update

© Donald F. Ferguson, 2021

REST and Isolation: Read then Update Conflict



- If
  - The written value
  - Depends on the read value.
  - The resource may be in the wrong state at the end.
- The execution was RRWW
- But the intent was RWRW.

© Donald F. Ferguson, 2021

Isolation/Concurrency Control

This requires conversation state, which would violate the REST Stateless principle.

- There are two basic approaches to implementing isolation
  - Locking/Pessimistic, e.g. cursor isolation
  - Optimistic: Before committing, each transaction verifies that no other transaction has modified the data it has read.
- How does this work in REST?
  - The server maintains an ETag (Entity Tag) for each resource.
  - Every time a resource's state changes, the server computes a new ETag.
  - The server includes the ETag in the header when returning data to the client.
  - The server may *optionally* maintain and return "Last Modified" time for resources.
- Semantics on updates
  - If-Match – value of a previous calls ETag response used with a PUT or DELETE. Server should only act if nobody else has modified the resource since you last fetched it. Otherwise provides a 412.
  - If-Modified-Since – value of a previous Last-Modified response used with a GET. Server should only provide a response if the resource was modified since the timestamp submitted. Use in conjunction with If-None-Match in case the change occurred within the same second. Otherwise provide a 304.
  - If-None-Match – value of a previous calls ETag response used with a GET. Server should only provide a response if the ETag doesn't match, i.e. the resource has been altered. Otherwise provide a 304.
  - If-Unmodified-Since – value of a previous Last-Modified response used with a PUT or DELETE. Server should only act if nobody else has modified the resource since you last fetched it. Otherwise provides a 412

© Donald F. Ferguson, 2021

Conditional Operations

What are DynamoDB Condition Expressions?

Before we learn the specifics of DynamoDB Condition Expressions, let's learn what they are and why you would want to use them.

A `ConditionExpression` is an optional parameter that you can use on write-based operations. If you include a Condition Expression in your write operation, it will be evaluated *prior to* executing the write. If the Condition Expression evaluates to false, the write will be aborted.

This can be useful in a number of common patterns, such as:

- Ensuring uniqueness,
- Validating business rules, and
- Confirming existance.

By using Condition Expressions, you can reduce the number of trips you make to DynamoDB and avoid race conditions when multiple clients are writing to DynamoDB at the same time.

By using Condition Expressions, you can reduce the number of trips you make to DynamoDB and avoid race conditions when multiple clients are writing to DynamoDB at the same time.

Condition Expressions can be used in the following write-based API operations:

- PutItem
- UpdateItem
- DeleteItem
- TransactWriteItems

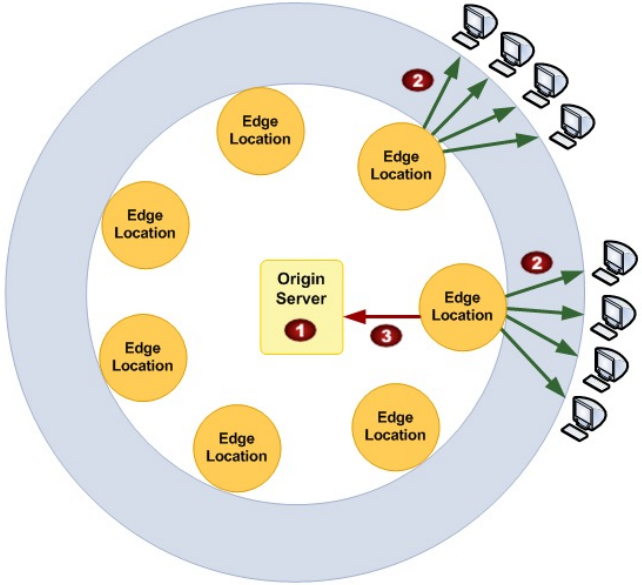
© Donald F. Ferguson, 2021

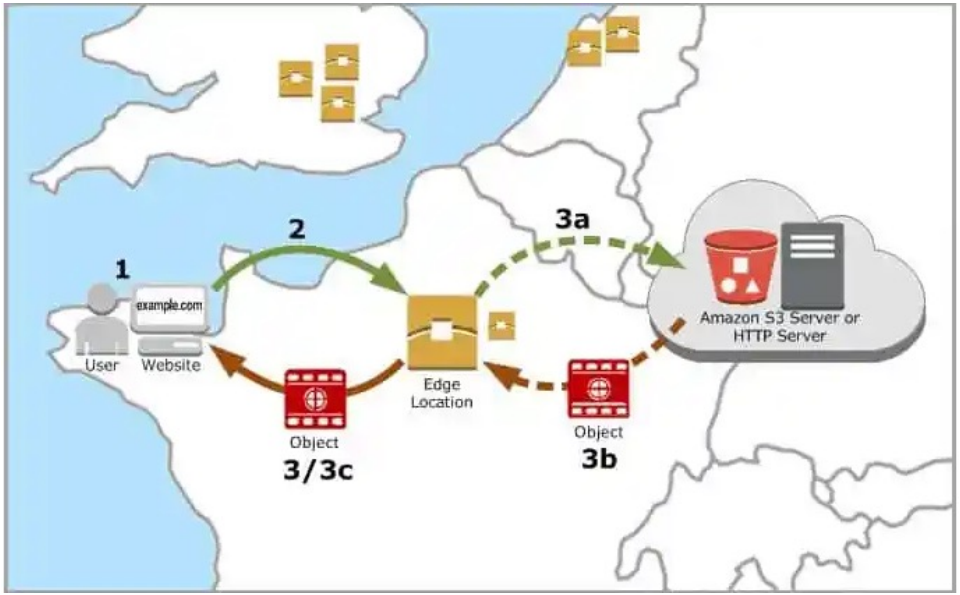
Hands-On CloudFront
DNS, Certificate, HTTPS

© Donald F. Ferguson, 2021

CloudFront

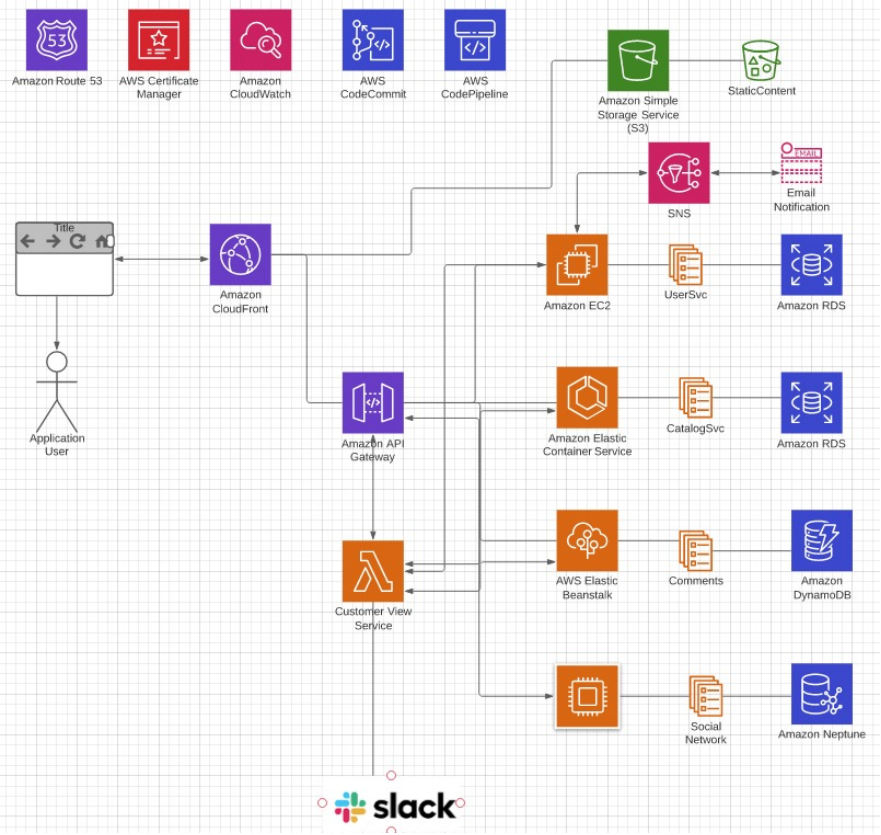
"A content delivery network, or content distribution network (CDN), is a geographically distributed network of proxy servers and their data centers. The goal is to provide high availability and performance by distributing the service spatially relative to end users." (https://en.wikipedia.org/wiki/Content_delivery_network)





© Donald F. Ferguson, 2021

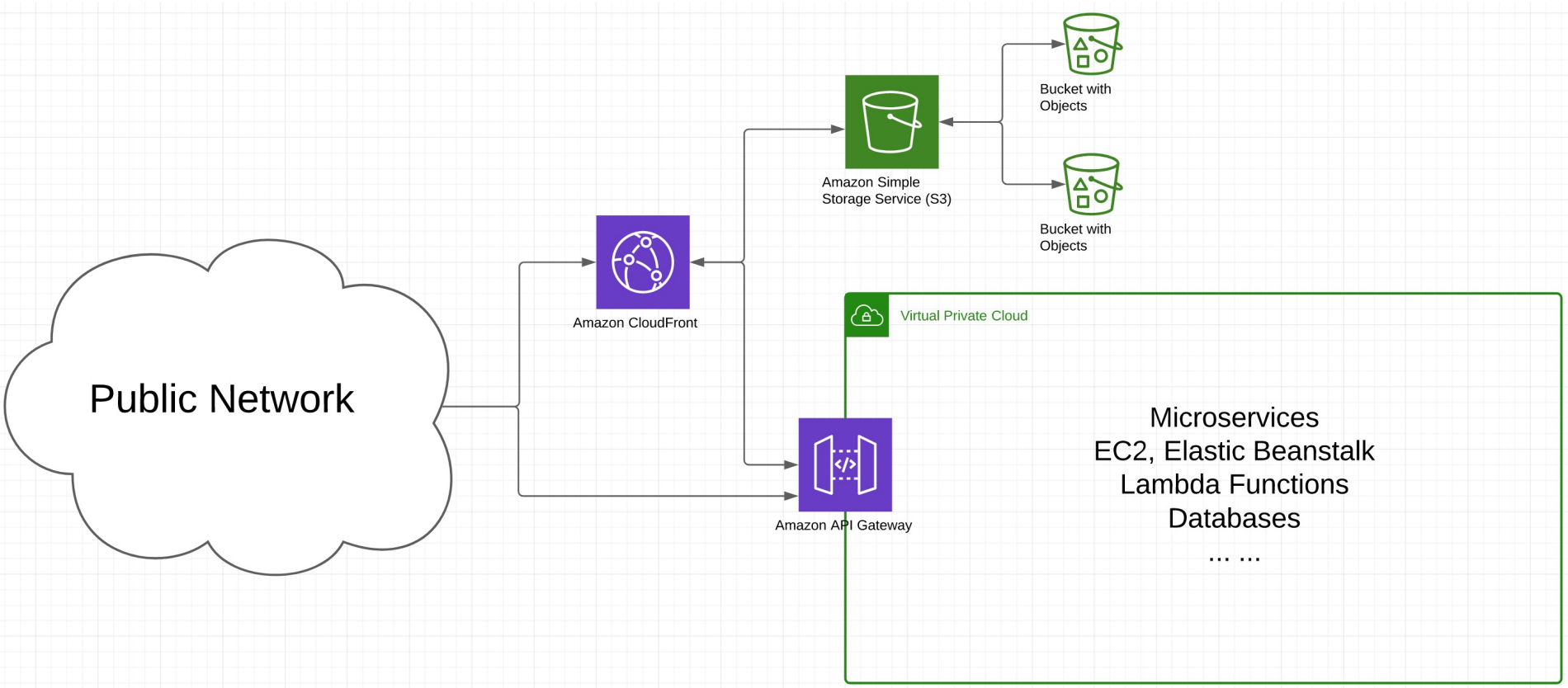
Overall Architecture



- There may be multiple
  - Microservices
  - Web UI projects/content
- CloudFront
  - Multiple S3 content → one site.
  - Forward API requests to API GW
- API GW manages/monitors APIs
- Networking/HTTPS requires
  - Domain, DNS
  - Certificates

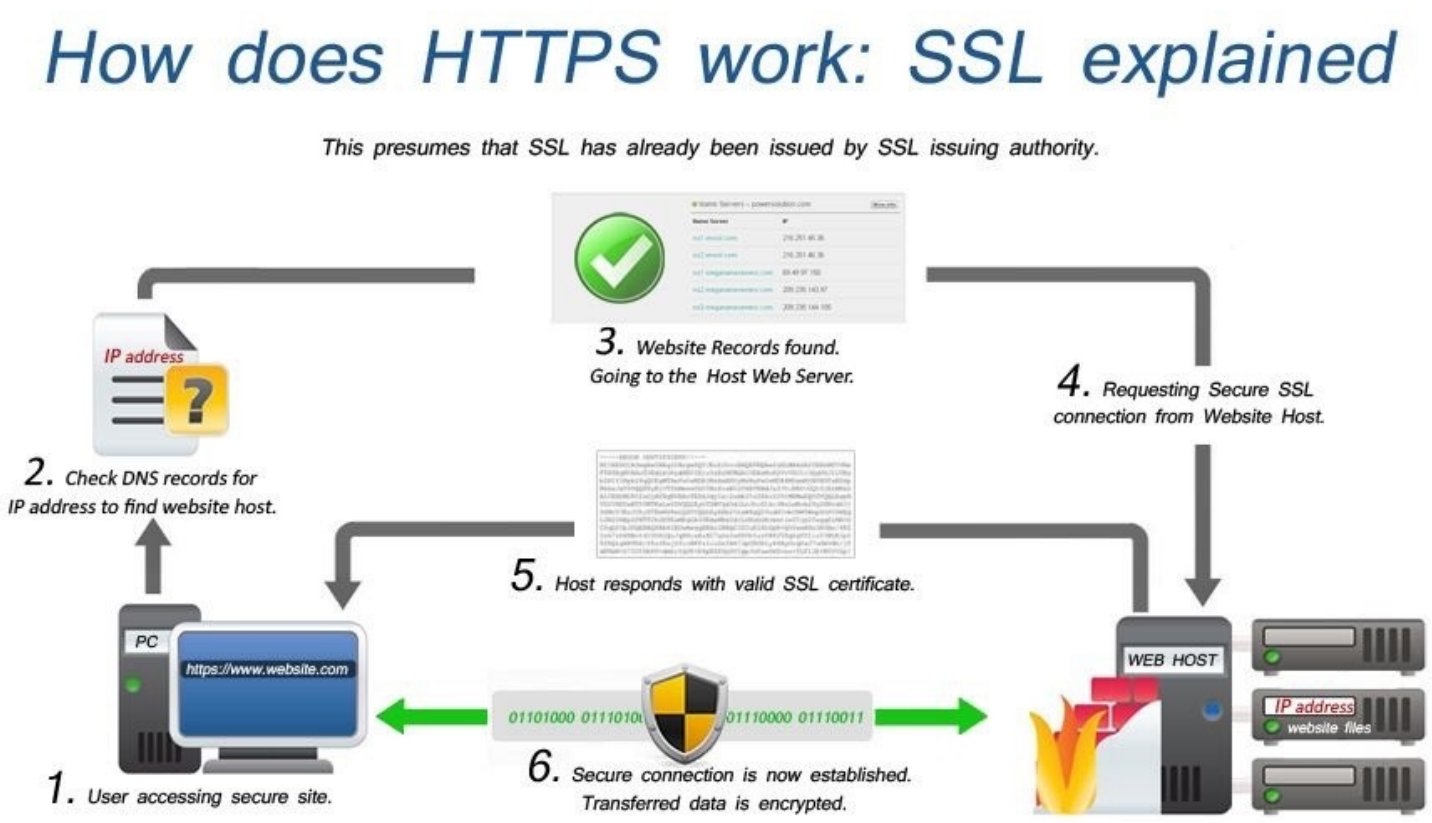
© Donald F. Ferguson, 2021

Big Picture



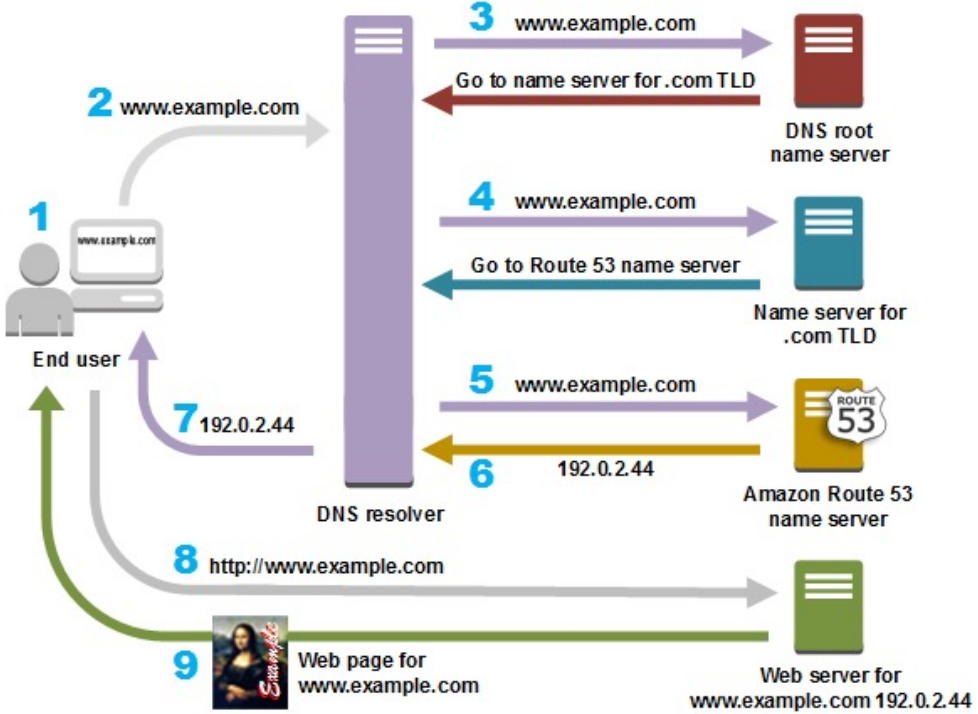
© Donald F. Ferguson, 2021

Certificates and HTTPS



© Donald F. Ferguson, 2021

DNS



© Donald F. Ferguson, 2021

Things to Demo

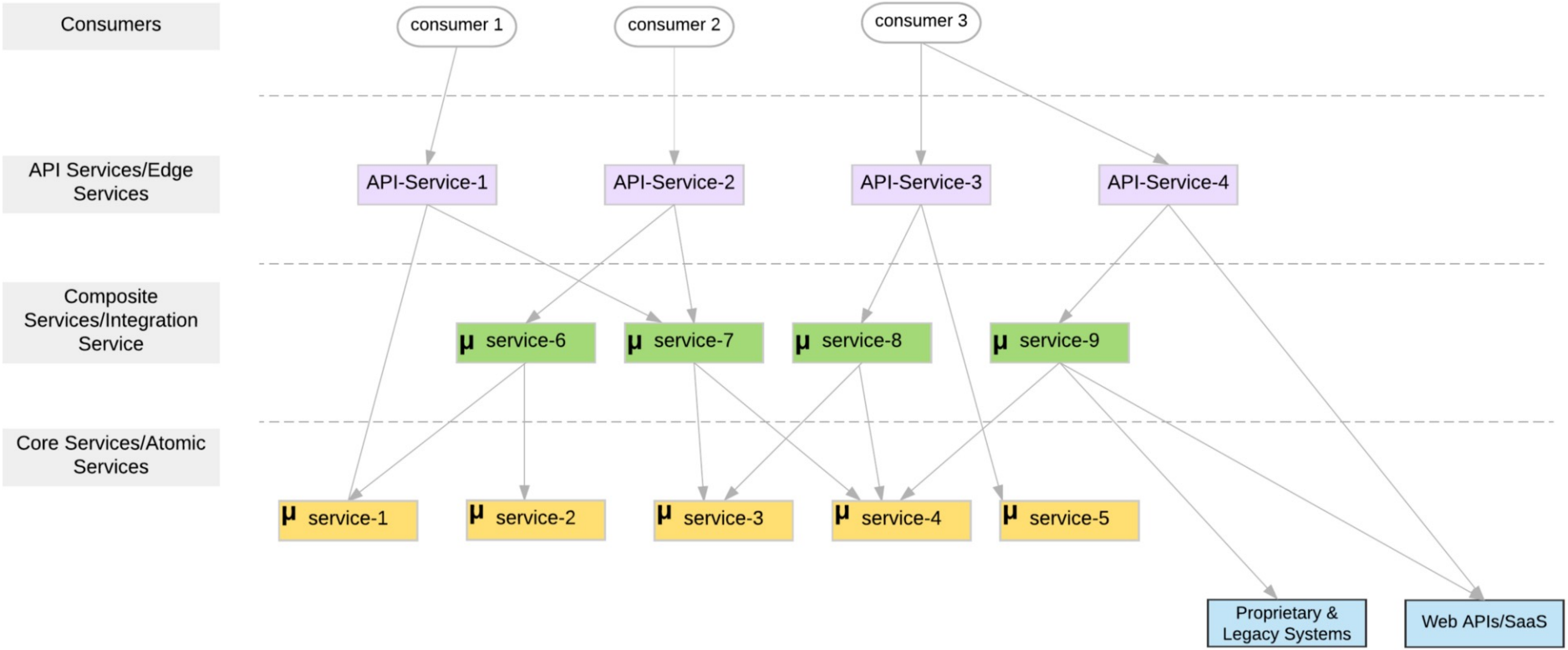
- CloudFront:
  - Distributions, Origins, Behavior
  - Monitoring, caching, alarms, ...
- Certificates
  - AWS implementation
  - Use in CloudFront and elsewhere
  - Visibility in browser, HTTPs
- Route 53
  - Proving ownership
  - Records

© Donald F. Ferguson, 2021

Service Composition

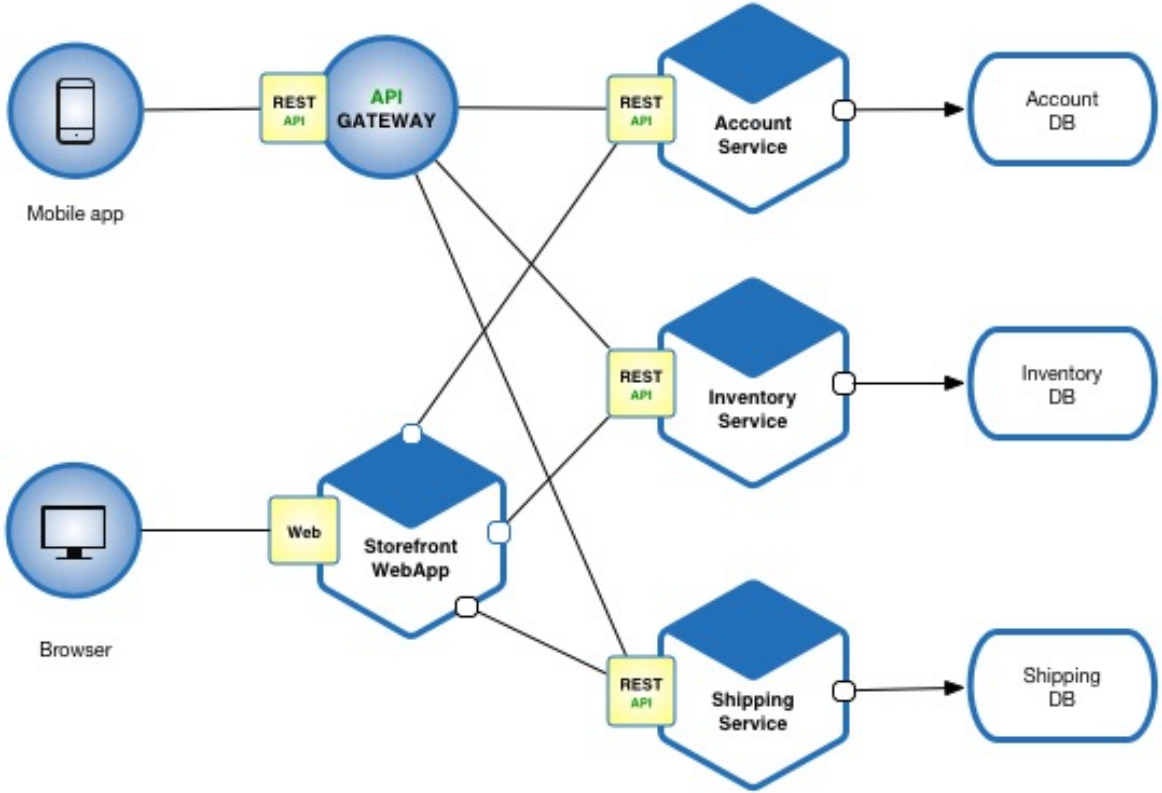
© Donald F. Ferguson, 2021

Microservice Layers



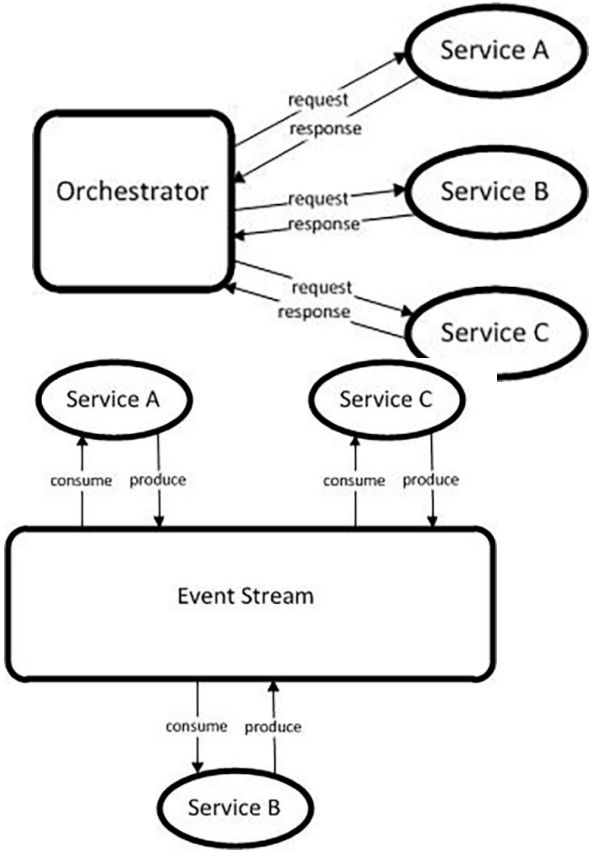
© Donald F. Ferguson, 2021

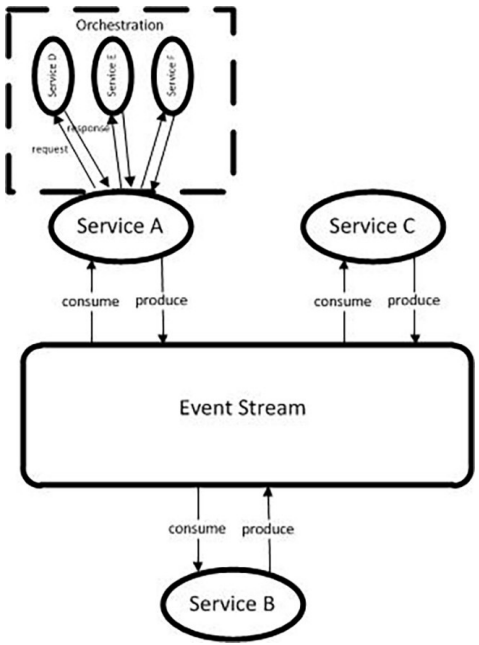
Another View

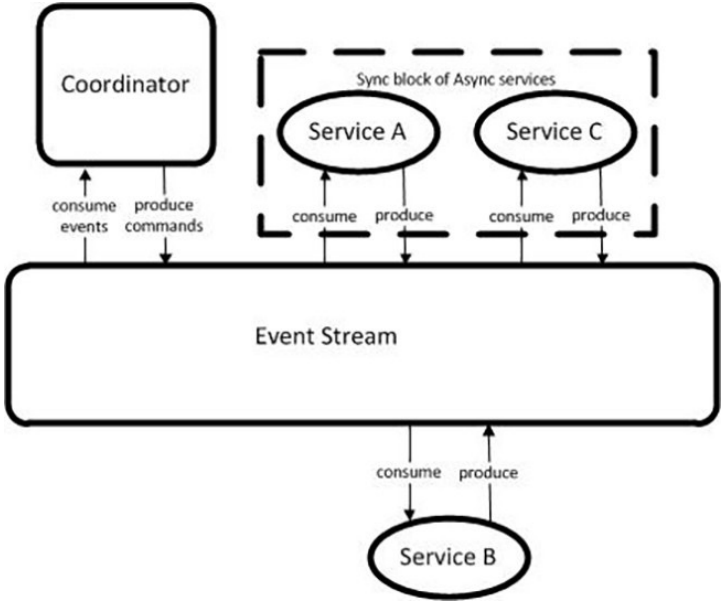


© Donald F. Ferguson, 2021

Models (https://medium.com/capital-one-tech/microservices-when-to-react-vs-orchestrate-c6b18308a14c)







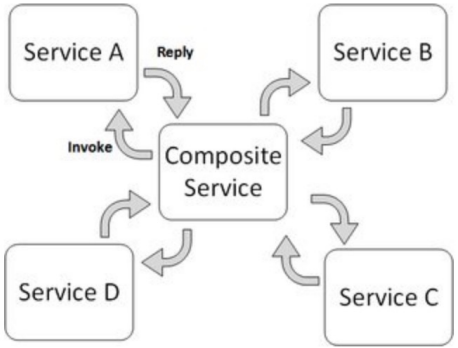
© Donald F. Ferguson, 2021

Concepts

Service orchestration

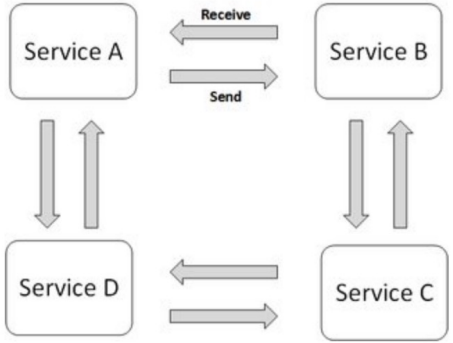
Service orchestration represents a single centralized executable business process (the orchestrator) that coordinates the interaction among different services. The orchestrator is responsible for invoking and combining the services.

The relationship between all the participating services are described by a single endpoint (i.e the composite service). The orchestration includes the management of transactions betweer individual services. Orchestration employs a centralized approach for service composition.



Service Choreography

Service choreography is a global description of the participating services, which is defined by exchange of messages, rules of interaction and agreements between two or more endpoints. Choreography employs a decentralized approach for service composition.

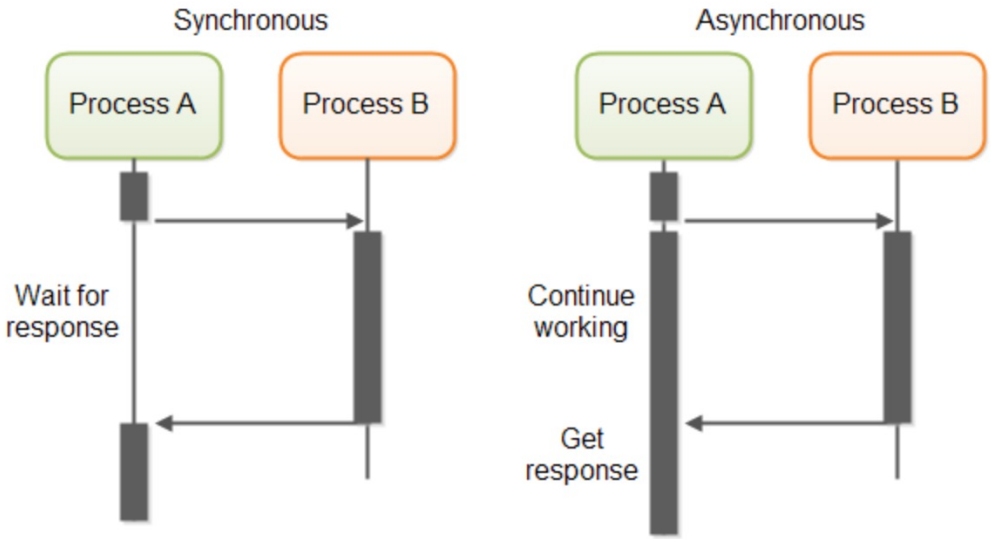


The choreography describes the interactions between multiple services, where as orchestration represents control from one party's perspective. This means that a **choreography** *differs* from an **orchestration** with respect to where the logic that controls the interactions between the services involved should reside.

These are not formally, rigorously defined terms.

© Donald F. Ferguson, 2021

Service Orchestration/Composition

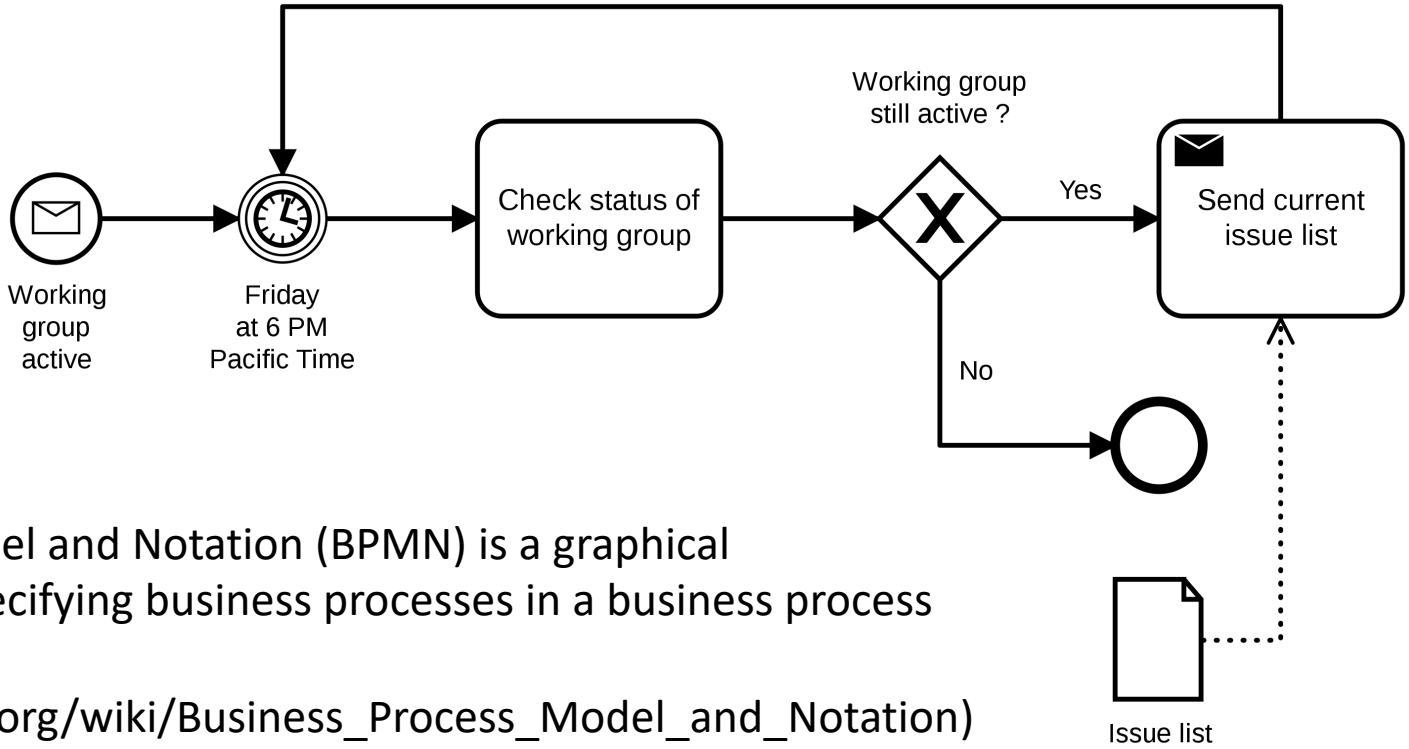


RESTFul HTTP calls can be implemented in both a synchronous and asynchronous fashion at an IO level.

- One call to a composite service
  - May drive calls to multiple other services.
  - Which in turn, may drive multiple calls to other services.
- Synchronous (Wait) and calling one at a time, in order is
  - Inefficient
  - Fragile
  - … …
- Asynchronous has benefits but can result in complex code.

© Donald F. Ferguson, 2021

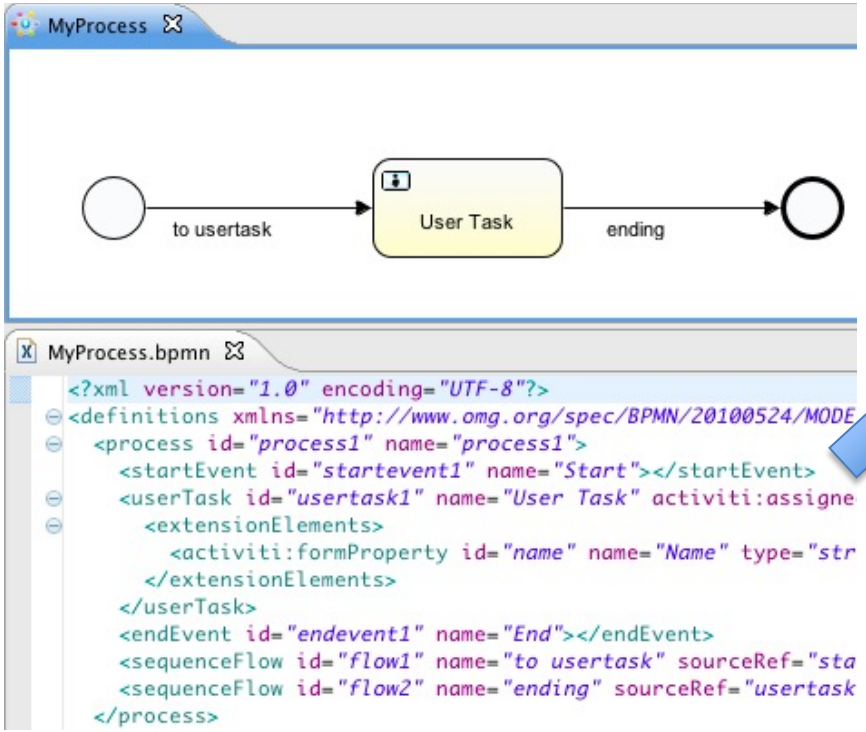
BPMN

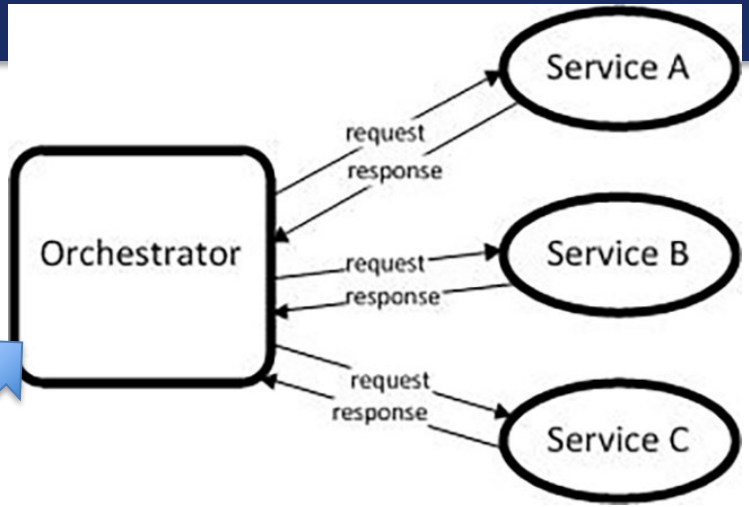


Business Process Model and Notation (BPMN) is a graphical representation for specifying business processes in a business process model.
(https://en.wikipedia.org/wiki/Business_Process_Model_and_Notation)

© Donald F. Ferguson, 2021

Executable Workflow Models

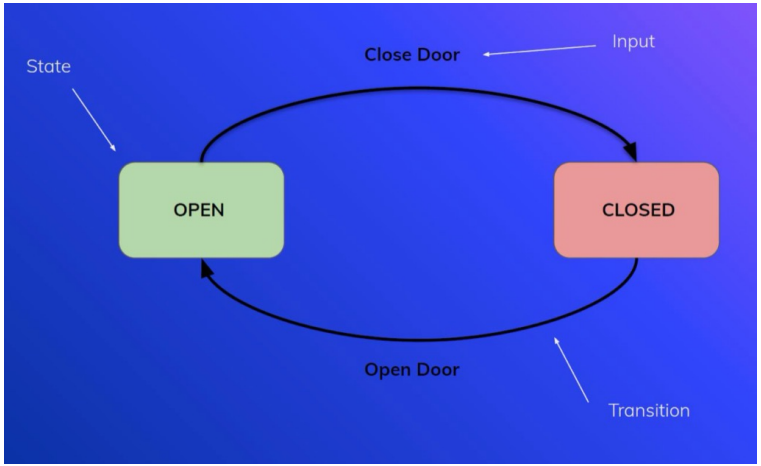


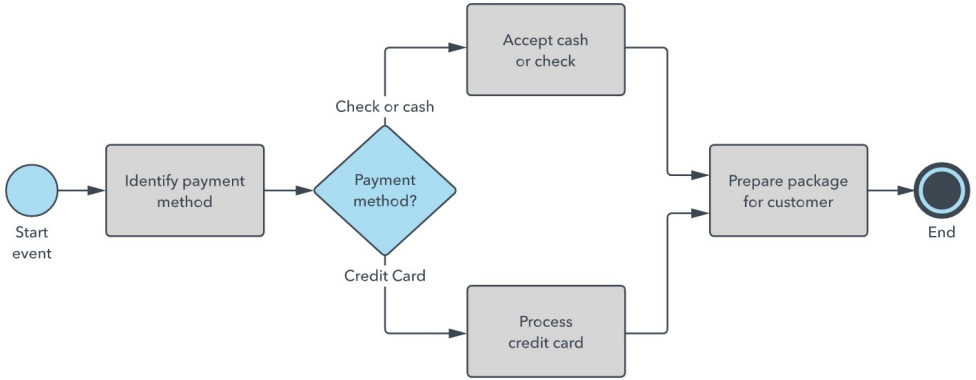


- The modeler is a visual editor for the underlying language.
- The "engine" can interpret the "program" to perform long running, asynchronous orchestration.

© Donald F. Ferguson, 2021

Two Core Models: State Machine, Control Graph





- The two models/approaches are equivalent.
- Workflow (BPMN) represents control flow, decisions, ... The state is:
  - The current active tasks.
  - The working memory of documents sent and receive.
  - There is a clear concept of “next tasks.”
- State machine has the concepts of (state, enabled events, action, next state).

© Donald F. Ferguson, 2021

AWS Step Functions

AWS Step Functions is a low-code visual workflow service used to orchestrate AWS services, automate business processes, and build serverless applications. Workflows manage failures, retries, parallelization, service integrations, and observability so developers can focus on higher-value business logic.

4,000 state transitions free

per month with the AWS Free Tier

Get started for free »

Benefits

Build and deploy rapidly

Get started with a simple drag-and-drop interface. With Step Functions, you can express complex business logic as low-code, event-driven workflows that connect services, systems or people within minutes.

Write less integration code

Compose AWS resources including Lambda, ECS, Fargate, Batch, DynamoDB, SNS, SQS, SageMaker, EventBridge, or EMR into resilient business workflows, data pipelines, or applications.

Build fault-tolerant and stateful workflows

Step Functions manages state, checkpoints, and restarts for you to make sure that your workflows execute in order and as expected. Built-in try/catch, retry, and rollback capabilities deal with errors and exceptions automatically based on your defined business logic.

Designed for any use case

Step Functions offers two workflow types - Standard or Express - that can be used depending on your specific use case. Standard Workflows are used to manage long-running workloads. Express Workflows support high-volume event processing workloads.

© Donald F. Ferguson, 2021

AWS Step Function Concepts

State Machine

States
- Task
- Choice
- Succeed/Fail
- Pass
- Wait
- Parallel

Transitions

Executions

You define the State Machine with the Amazon States Language

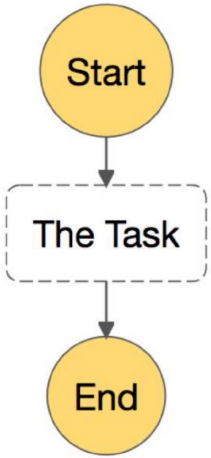
A diagram will be generated based on that structure

@technovangelist

© Donald F. Ferguson, 2021

State Example: Task

```
{
  "Comment": "A simple task example",
  "StartAt": "The Task",
  "States": {
    "The Task": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:REGION:ACCOUNT_ID:function:AwesomeTask",
      "End": true
    }
  }
}
```

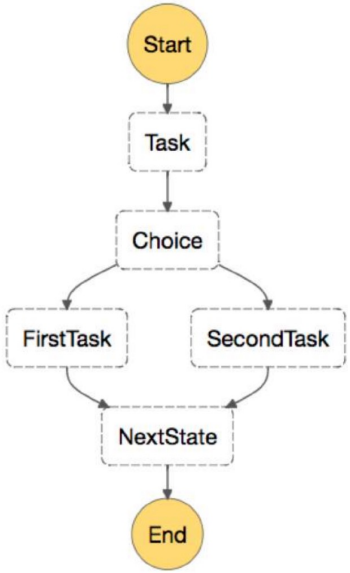


@technovangelist

© Donald F. Ferguson, 2021

State Example: Choice

```
10    "Choice": {
11      "Type" : "Choice",
12      "Choices": [
13        {
14          "Variable": "$.foo",
15          "NumericEquals": 1,
16          "Next": "FirstTask"
17        },
18        {
19          "Variable": "$.foo",
20          "NumericEquals": 2,
21          "Next": "SecondTask"
22        }
23      ]
24    },
```

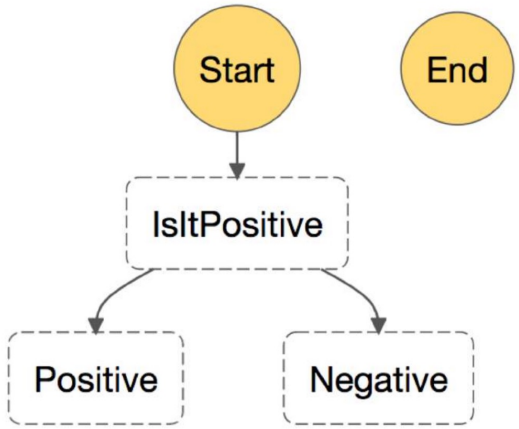


@technovangelist

© Donald F. Ferguson, 2021

State Example: Succeed / Fail

```
7    "Choices": [
8      {
9        "Variable": "$.foo",
10       "NumericGreaterThan": 0,
11       "Next": "Positive"
12     }
13   ],
14   "Default": "Negative"
15 },
16
17 "Positive": {
18   "Type" : "Succeed"
19 },
20
21 "Negative": {
22   "Type": "Fail",
23   "Error": "DefaultStateError",
24   "Cause": "Its not positive!"
25 }
```

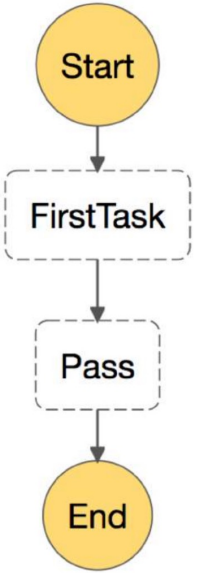


@technovangelist

© Donald F. Ferguson, 2021

State Example: Pass

```
{
  "Comment": "A simple pass",
  "StartAt": "FirstTask",
  "States": {
    "FirstTask": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:REGION:ACCOUNT_ID:function:AwesomeTask",
      "Next": "Pass"
    },
    "Pass": {
      "Type": "Pass",
      "Result": "Wow, this was exciting",
      "ResultPath": "ExtraDetail",
      "End": true
    }
  }
}
```

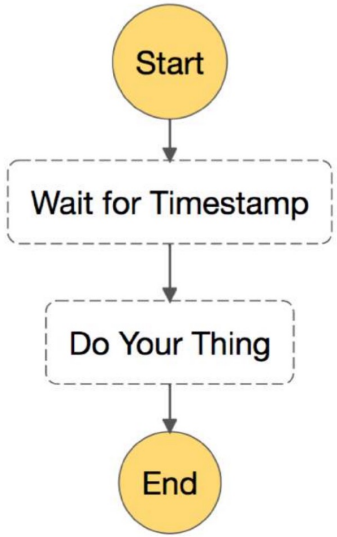


@technovangelist

© Donald F. Ferguson, 2021

State Example: Wait

```
{
  "Comment": "A simple wait example",
  "StartAt": "Wait for Timestamp",
  "States": {
    "Wait for Timestamp": {
      "Type": "Wait",
      "TimestampPath": "$.trigger_date",
      "Next": "Do Your Thing"
    },
    "Do Your Thing": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:REGION:ACCOUNT_ID:function:DoTheThing",
      "End": true
    }
  }
}
```

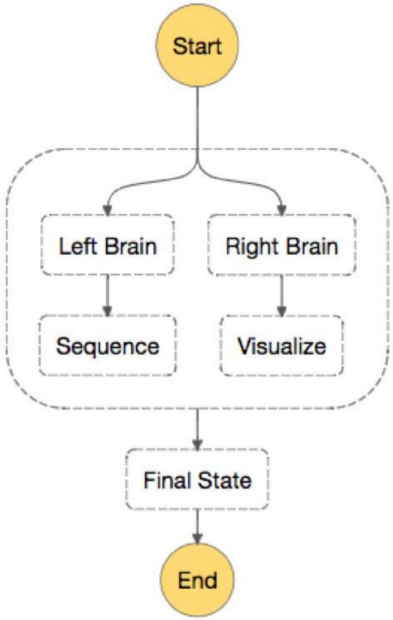


@technovangelist

© Donald F. Ferguson, 2021

State Example: Parallel

```
"Parallel": {
  "Type": "Parallel",
  "Next": "Final State",
  "Branches": [
    {
      "StartAt": "Left Brain",
      "States": {
        "Left Brain": {
          "Type": "Wait",
          "Seconds": 2,
          "Next": "Sequence"
        },
        "Sequence": {
          "Type": "Task",
          "Resource": "arn:aws:lambda:REGION:ACCOUNT_ID:function:sequence",
          "End": true
        }
      }
    },
    {
      "StartAt": "Right Brain",
      "States": {
        "Right Brain": {
          "Type": "Pass",
          "Next": "Visualize"
```



@technovangelist

© Donald F. Ferguson, 2021

Next Steps

- Flesh out and expand
  - CloudFront
  - S3 static content
  - API gateway
  - Microservices
- Develop composite microservices and get a feel for the patterns
  - Step Functions
  - Asynchronous Call-Return
  - Event Driven
  - I will provide some examples and you must adapt to your project.

© Donald F. Ferguson, 2021