

# COMS E6998: Microservices and Cloud Applications

*Lecture 5: Idempotency, Conditional Execution, Orchestration, Pub/Sub, API GW*

Dr. Donald F. Ferguson  
dff9@columbia.edu

© Donald F. Ferguson, 2017. All rights reserved.

Questions?  
Comments?  
(Pagination)

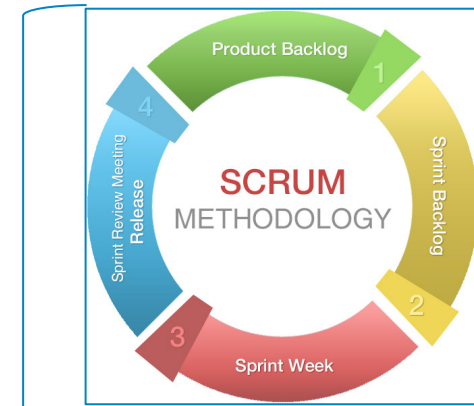
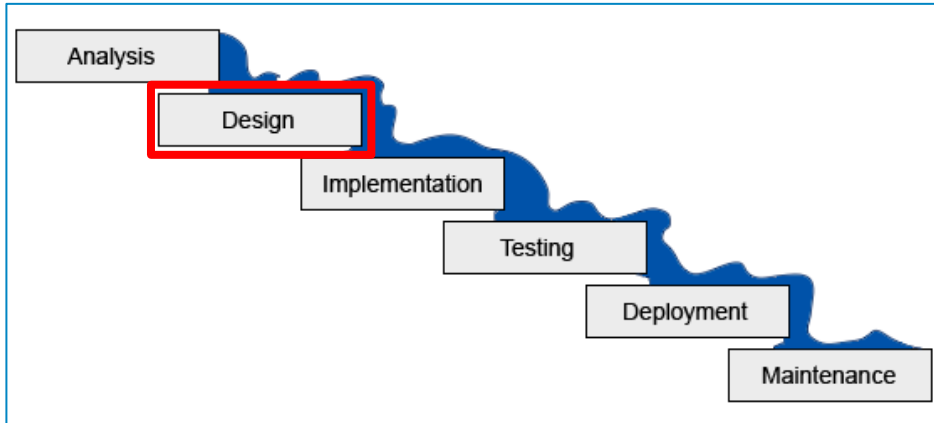
Development  
Methodology

Design Docs

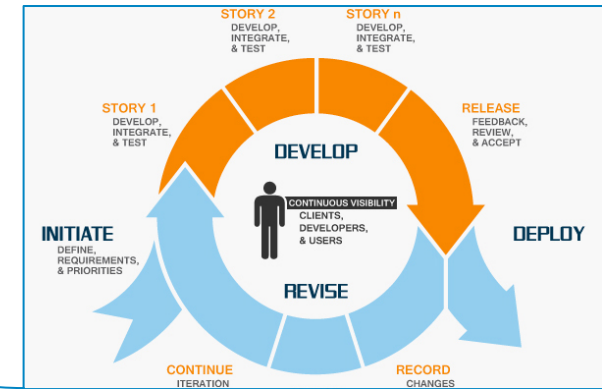
Project Submission  
Requirements

# Agile Development

## Traditional “Waterfall” Methodology



*Design seems to be missing?*



# Observations

Weinberg's Second Law: “If builders built buildings the way programmers wrote programs, then the first woodpecker that came along would destroy civilization.”

Sr. Celestina: “If you cannot write it down, you do not understand it.”

“it is a custom  
More honor'd in the breach than the observance”  
Hamlet Act 1, scene 4, 7–16

Agile Manifesto (<http://agilemanifesto.org/>):

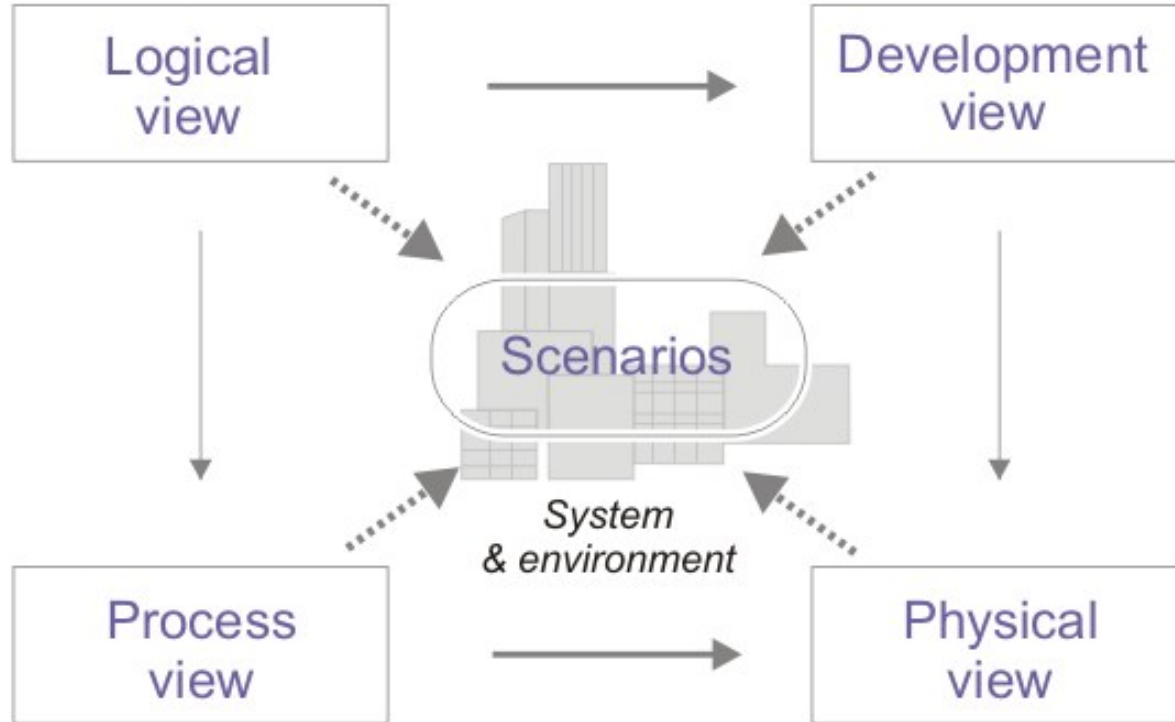
- Individuals and interactions over processes and tools
- Working software over **comprehensive** documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Also, I want to help you build your “soft skills.”

- Presentations
- Elevator Speeches.
- Project demos/reviews.

# 4+1 architectural view model

[http://en.wikipedia.org/wiki/4%2B1\\_architectural\\_view\\_model](http://en.wikipedia.org/wiki/4%2B1_architectural_view_model)



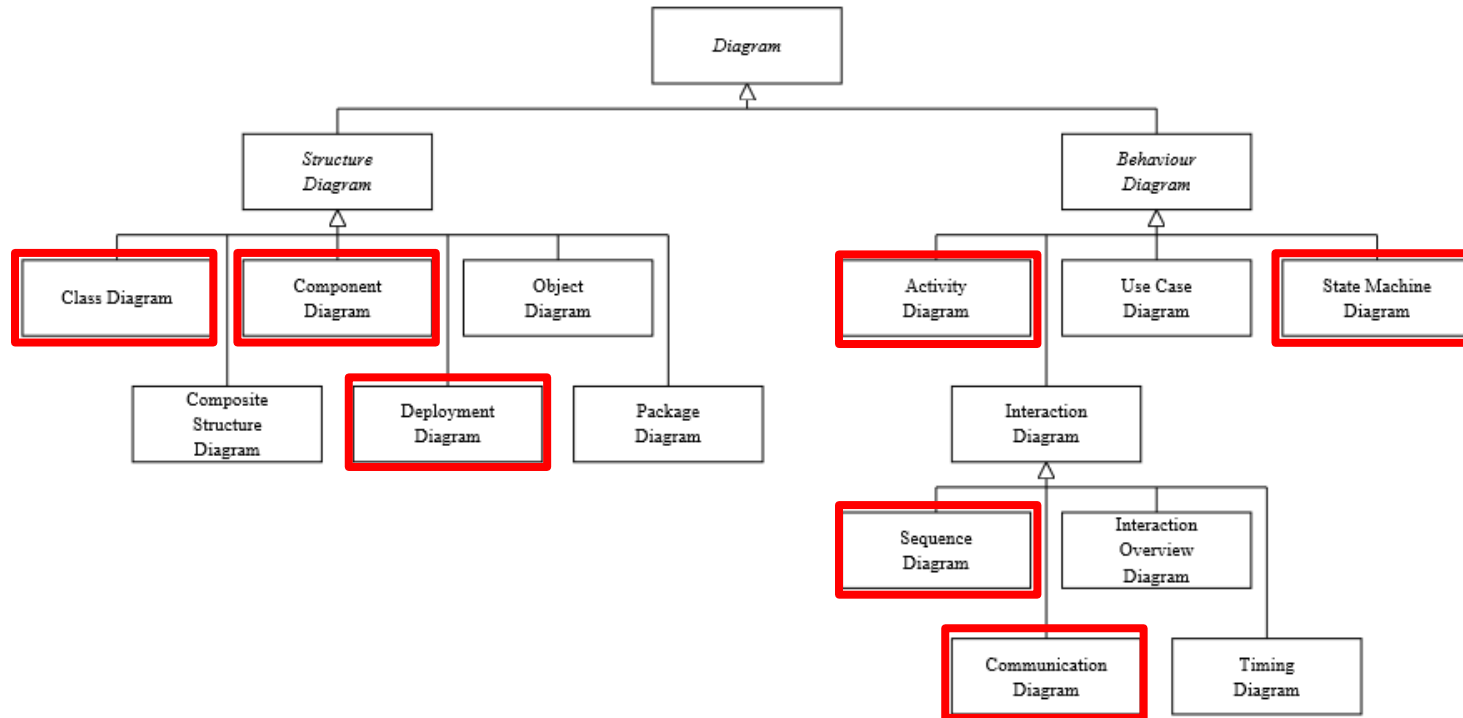
# The Architecture Views

[http://en.wikipedia.org/wiki/4%2B1\\_architectural\\_view\\_model](http://en.wikipedia.org/wiki/4%2B1_architectural_view_model)

- **Logical view** : The logical view is concerned with the functionality that the system provides to end-users. UML Diagrams used to represent the logical view include [Class diagram](#), [Communication diagram](#), [Sequence diagram](#).<sup>[2]</sup>
- **Development view** : The development view illustrates a system from a programmer's perspective and is concerned with software management. This view is also known as the implementation view. It uses the UML [Component diagram](#) to describe system components. UML Diagrams used to represent the development view include the [Package diagram](#).<sup>[2]</sup>
- **Process view** : The process view deals with the dynamic aspects of the system, explains the system processes and how they communicate, and focuses on the runtime behavior of the system. The process view addresses concurrency, distribution, integrators, performance, and scalability, etc. UML Diagrams to represent process view include the [Activity diagram](#).<sup>[2]</sup>
- **Physical view** : The physical view depicts the system from a system engineer's point of view. It is concerned with the topology of software components on the physical layer, as well as the physical connections between these components. This view is also known as the deployment view. UML Diagrams used to represent physical view include the [Deployment diagram](#).<sup>[2]</sup>
- **Scenarios** : The description of an architecture is illustrated using a small set of [use cases](#), or scenarios which become a fifth view. The scenarios describe sequences of interactions between objects, and between processes. They are used to identify architectural elements and to illustrate and validate the architecture design. They also serve as a starting point for tests of an architecture prototype. This view is **also known as use case view**

# Some Diagrams

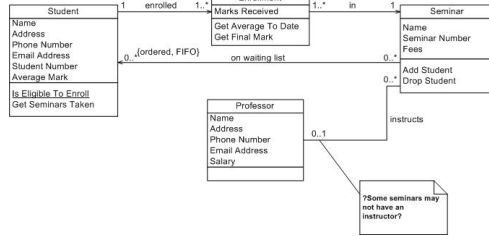
<http://www.agilemodeling.com/essays/umlDiagrams.htm>



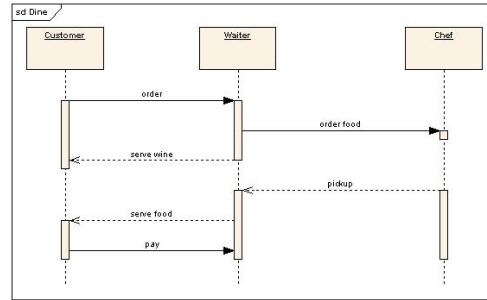


# Some Diagrams

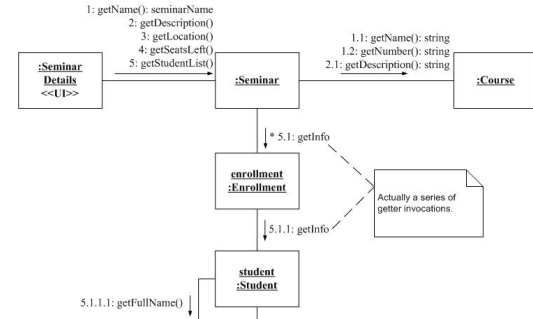
## Class Diagram



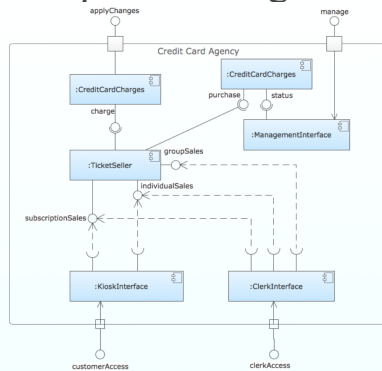
## Sequence Diagram



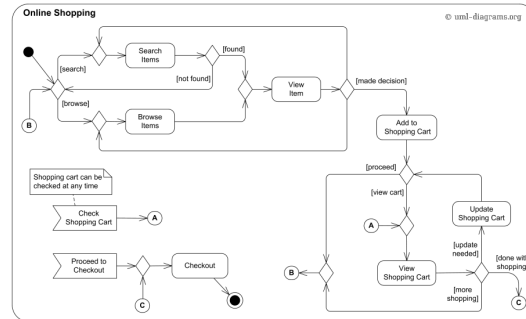
## Communication Diagram



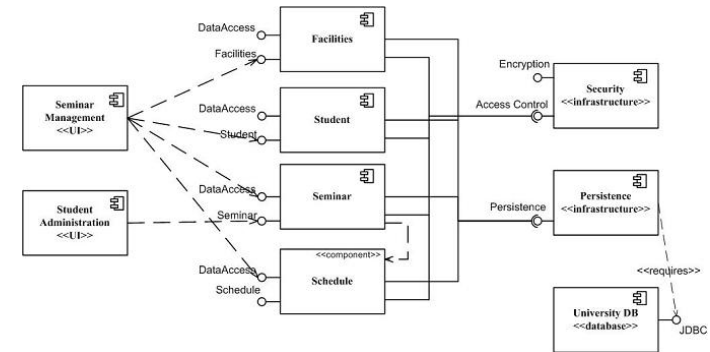
## Component Diagram



## Activity Diagram

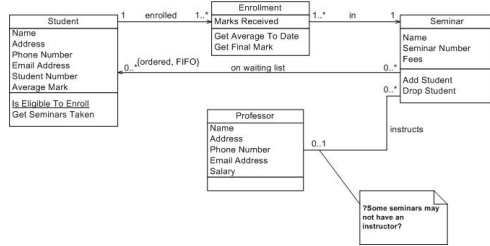


## Component Diagram

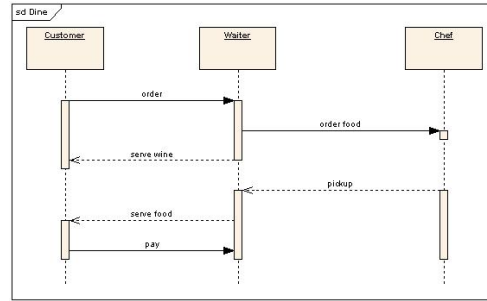


# Application to Project Submissions

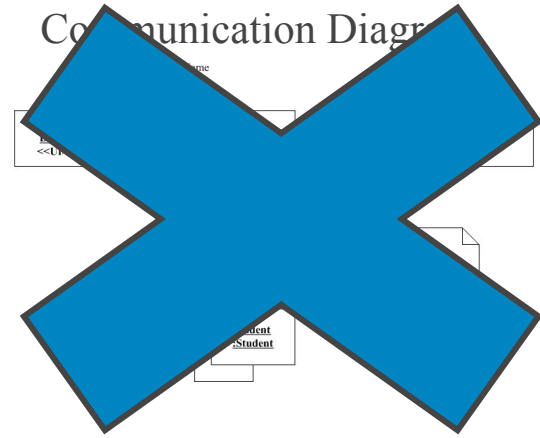
## Class Diagram



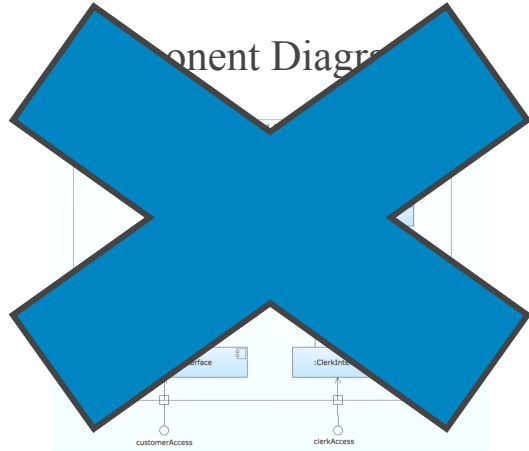
## Sequence Diagram



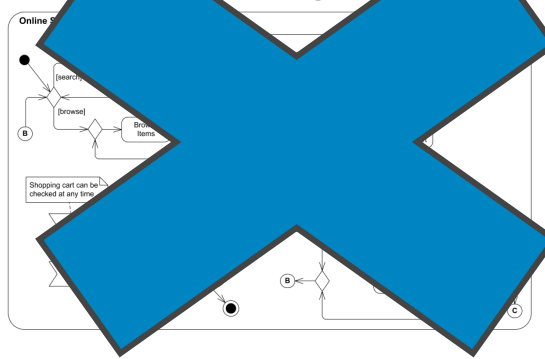
## Communication Diagram



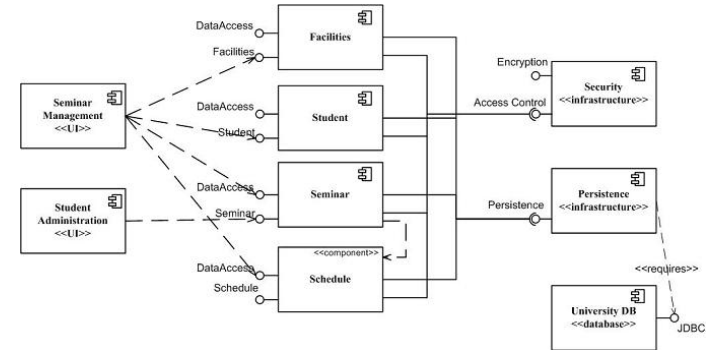
## Component Diagram



## Activity Diagram



## Component Diagram



# User Stories

- **User stories** are short, simple descriptions of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system. They typically follow a simple template: (<https://www.mountaingoatsoftware.com/agile/user-stories>)
  - *As a <type of user>, I want <some goal> so that <some reason>.*
- Example user stories (<http://www.agilemodeling.com/artifacts/userStory.htm>)
  - Students can purchase monthly parking passes online.
  - Parking passes can be paid via credit cards.
  - Parking passes can be paid via PayPal.
  - Professors can input student marks.
  - Students can obtain their current seminar schedule.
  - Students can order official transcripts.
  - Students can only enroll in seminars for which they have prerequisites.
  - Transcripts will be available online via a standard browser

# Agile Development and User Stories

## 2.3 User Stories

*You can either embed a document that contains the user stories; provide a link to an online system where the stories can be reviewed. If neither of those are viable, you can include them here in a similar format to the table below.*

Title	<Insert title here>
Description	<Describe use-case objectives here>
Actors/Roles Involved	
Pre-conditions	
Flow of Events	
Post-conditions	
Assumptions	
Limitations	

# Documenting API Example

(We will use Swagger and Swagger Editor)

URI	<code>https://mysite.com:3911/api/members/{id}</code>
HTTP verb	PUT
Parameters	<code>id</code> : Card number of the member.
Body	<code>name</code> : Name of the member. <code>email</code> : Email adress of the member. <code>langage</code> : Langage used by member (Fr_CA ou En_US)
Sample body	<pre>{   "name": "Mario Cardinal",   "email": "mcardinal@mariocardinal.com",   "language": "fr_CA" }</pre>
Success Response	Status Code: 204 No Content
Error Response	<b>Status Code:</b> 400 Bad Request, <b>Body:</b> {"Error Code": "..."} <b>Status Code:</b> 401 Unauthenticated, see <b>WWW-Authenticate</b> value in header <b>Status Code:</b> 403 Forbidden <b>Status Code:</b> 404 Not Found <b>Status Code:</b> 429 Too Many Requests, see <b>Retry-After</b> value in header <b>Status Code:</b> 500 Internal Server Error <b>Status Code:</b> 503 Service Unavailable
Error Code	<b>10:</b> Inactive member <b>20:</b> Denied access member <b>110:</b> Database issues, Retry later

# Documenting API Example

(We will use Swagger and Swagger Editor)

URI	https://mysite.com:3911/api/members/{id}
HTTP verb	PUT
Parameters	
Body	
Sample b	
Success Response	
Error Res	
Error Code	Status code: 503 Service Unavailable 10: Inactive member 20: Denied access member 110: Database issues, Retry later

Use Swagger  
to document APIs.

# Top-Level Design Specification

- Comments
  - I have presented a very *systematic* version of TLDS content.
  - All development projects should do *some flavor* of this approach, ideally simple and living/dynamic.
  - Just want to give you a feel for the types of things you can include in your TLDS, but you only need a really simple version.
  - Use common sense.
- For your TLDS
  - Definitely define, prioritize and implement simple user stories.
  - You do not need to use UML, but should use some of the concepts.
  - Some useful diagrams/artifacts
    - Swagger API document.
    - Component diagram.
    - Class diagram documenting the logical datamodels/ resource models for components.
    - Some sequence diagrams showing end-to-end flows and Lambda functions in flows.
- Why am I asking you to do this?
  - Documenting and presenting user stories, APIs and designs is a crucial career skill.
  - Learning the basic methodology is essential to converting unclear business requirements into a solution that satisfies both engineering and business stakeholders.
  - *I purposely give you unclear specifications to force you to THINK and apply methodology.*
  - The grade is based on the “correct answer,” but because the request is unclear, you use the methodology to define *your correct answer*.

# API Management API Gateway



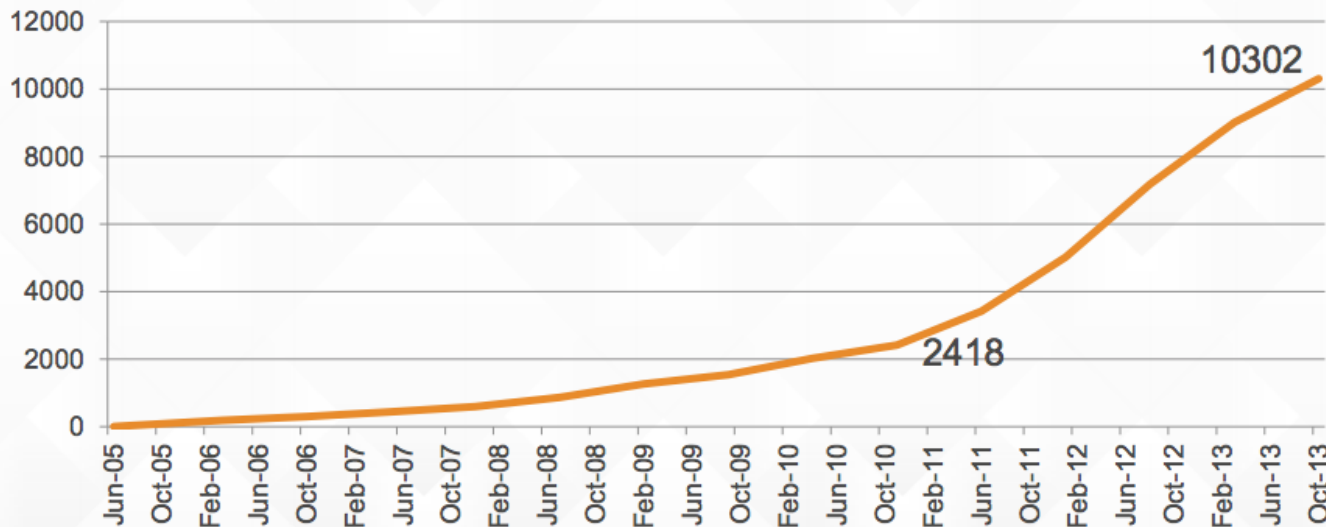
# API Management

# API Proliferation

(<https://www.slideshare.net/AmazonWebServices/build-and-manage-your-apis-with-amazon-api-gateway>)

## API proliferation

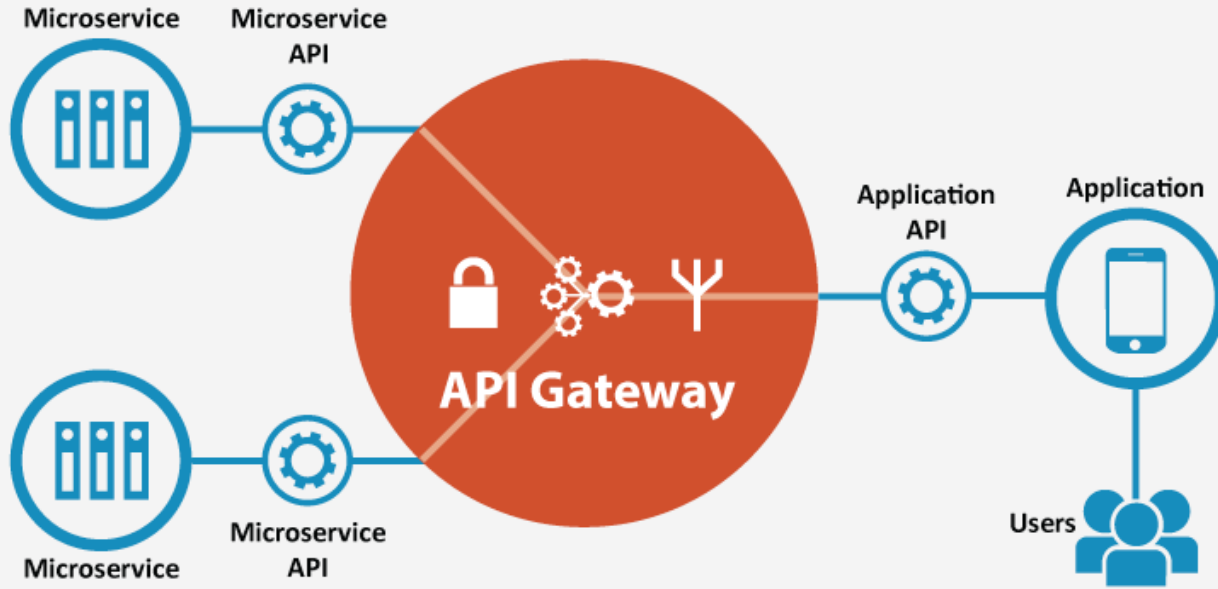
The number of published APIs is growing rapidly



\* Data from ProgrammableWeb



# API Gateway/API Management/API Strategy



In successful enterprise microservice architectures, API gateways are commonly used to deliver three vital functions:

- Security
- Orchestration
- Routing

<http://www.apiacademy.co/>

## Goals of API Design

Nevertheless, many API programs are growing out of previous SOA initiatives. Web services focused on internal or partner integrations are being opened up to developers—both within and outside the enterprise. During this process, it is important for API designers to remember that an API program has drivers and requirements quite different from the ones that initially led enterprises to open their IT assets via Web services.

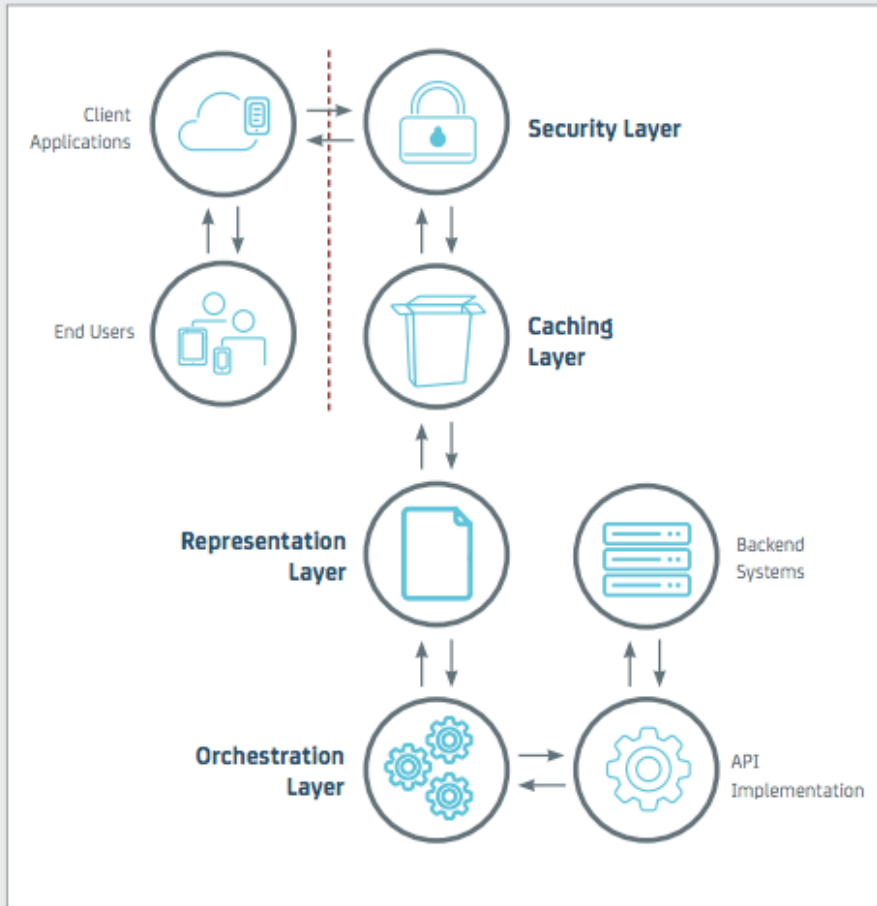
With this in mind, the broad goals of API design in general can be defined as:

- Enabling self-service for app developers and app users alike
- Reducing barriers to accessing valuable enterprise resources
- Prioritizing the needs and preferences of client app developers
- Encouraging collaboration between and among internal and external resources
- Addressing the security and scaling issues of exposing IT assets to the open market

Above all, API design must be focused on maximizing the business value of the interface. In part two, we will take a closer look at how APIs add value to the business.

# Architecture Layer

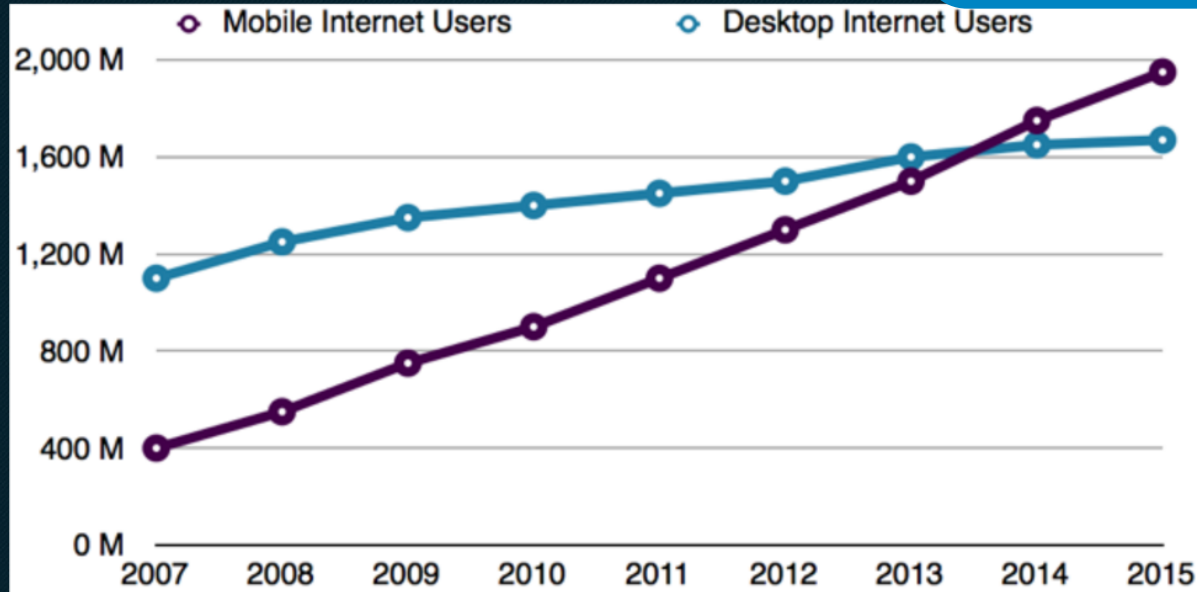
Figure 6: Architectural Layers



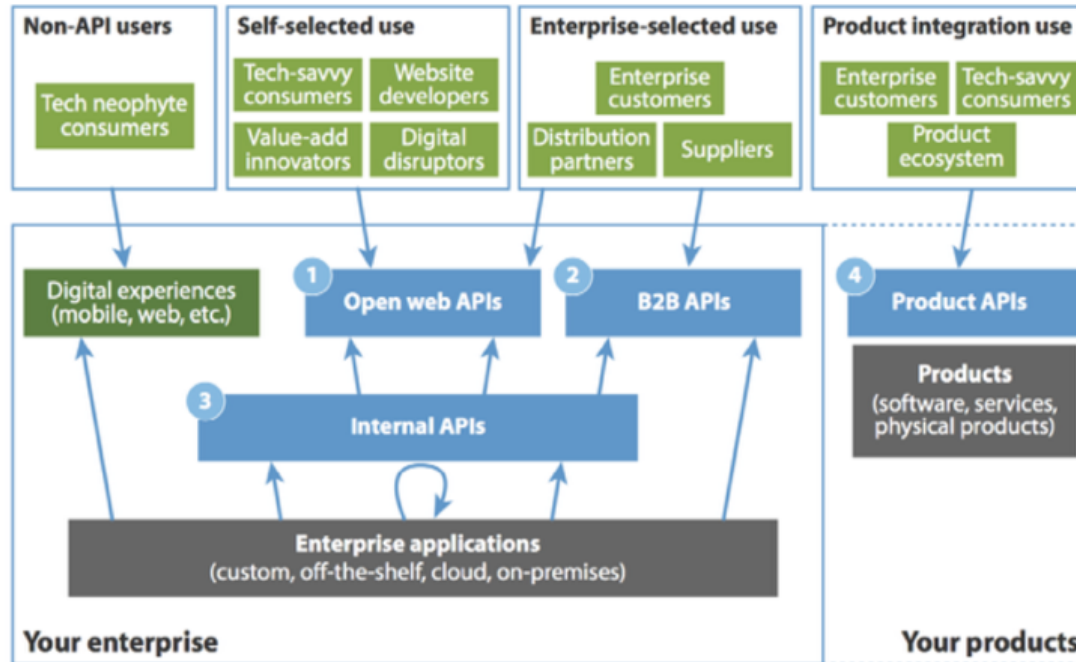
## DEMAND

Changing Consumer Expectations

Enable apps other than your client to access your APIs and drive business/value.

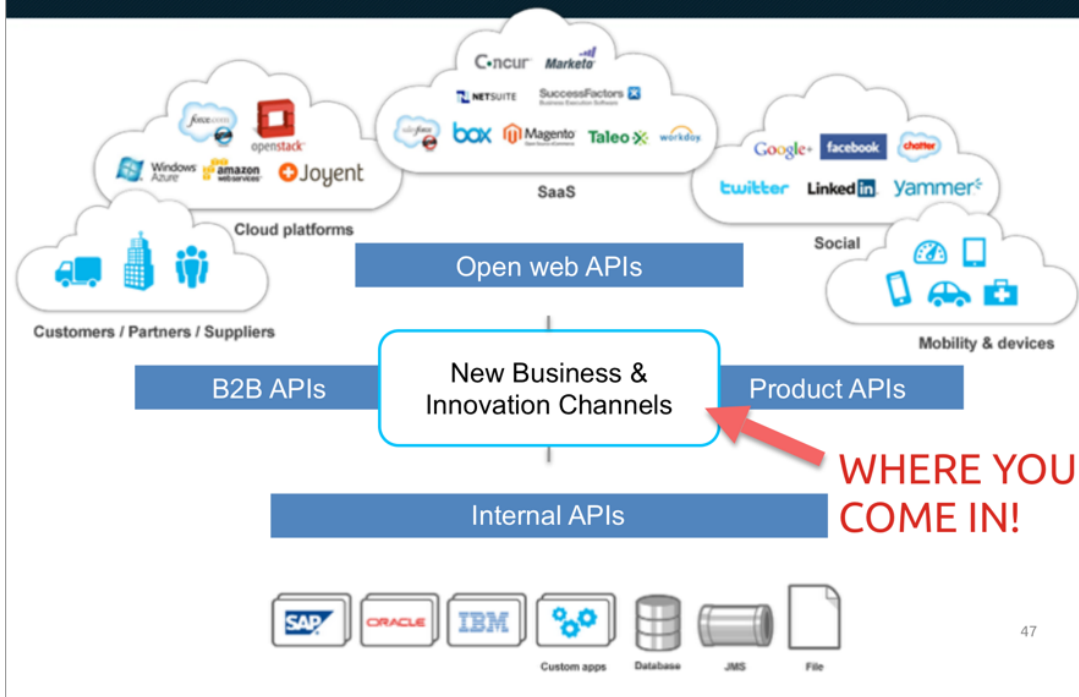


# What Kinds of APIs are in the New Enterprise



Source: Forrester, Establish your API design, June 2013

# API models in the Enterprise



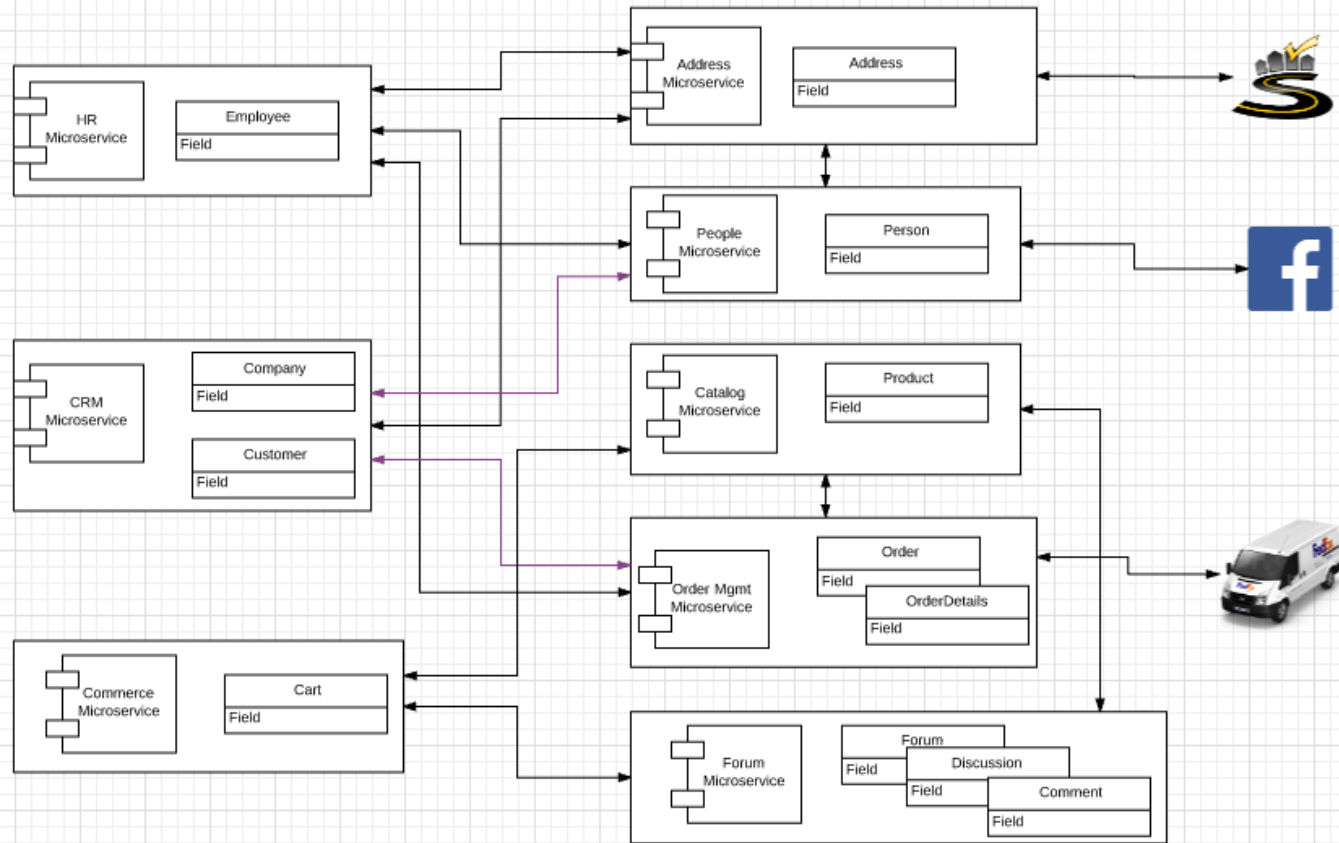
Much of this appears to be independent of the cloud and – as-a-Service, but ...

- Internal is a statement of ownership, not deployment.
- The gateway/boundary
  - Is logical.
  - May be a cloud service.
- API economy is about
  - Finding APIs.
  - Consuming APIs.
  - Composing APIs.
  - Publishing APIs.



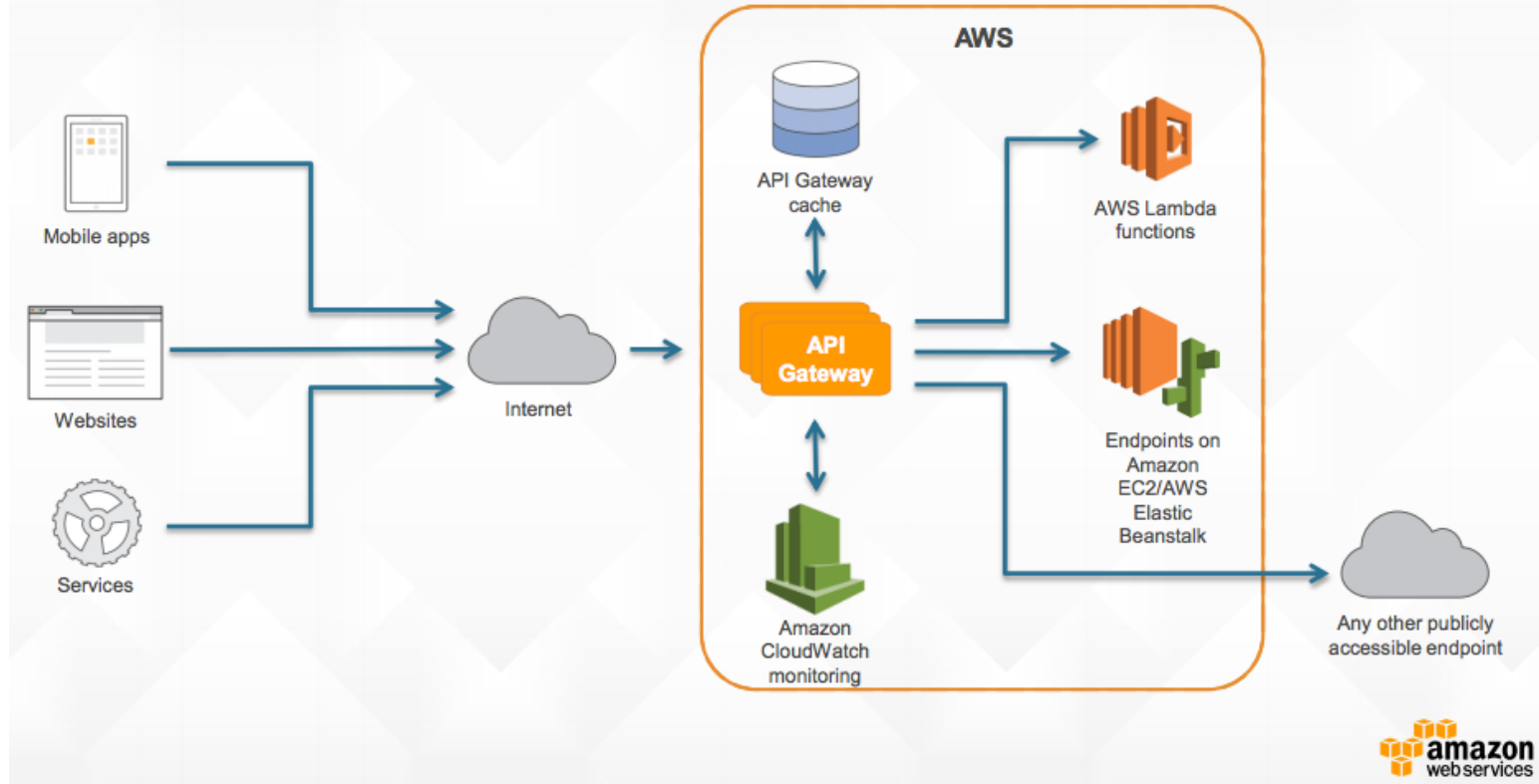
# API Gateway

# Microservice Model

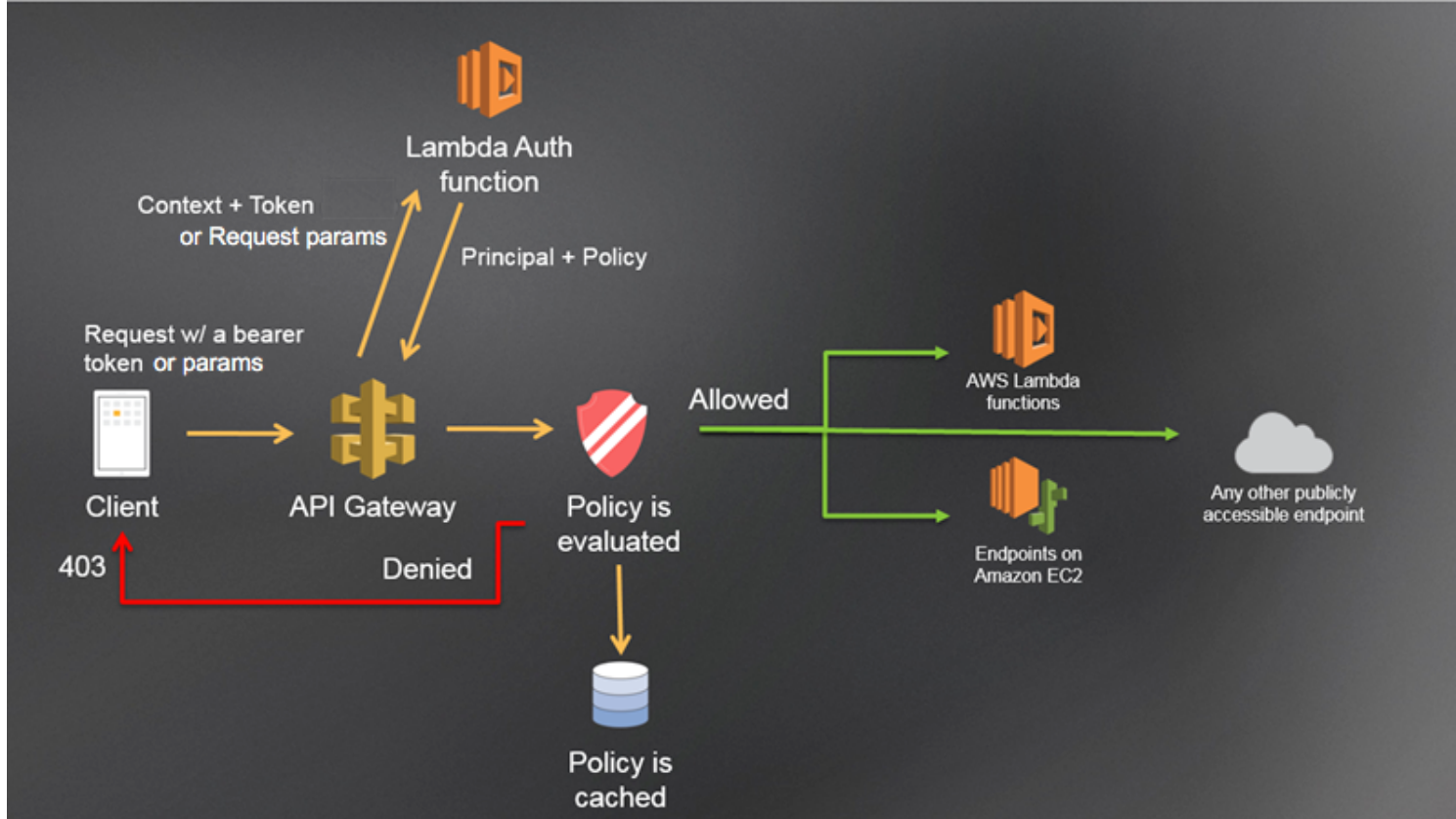


# API Gateway

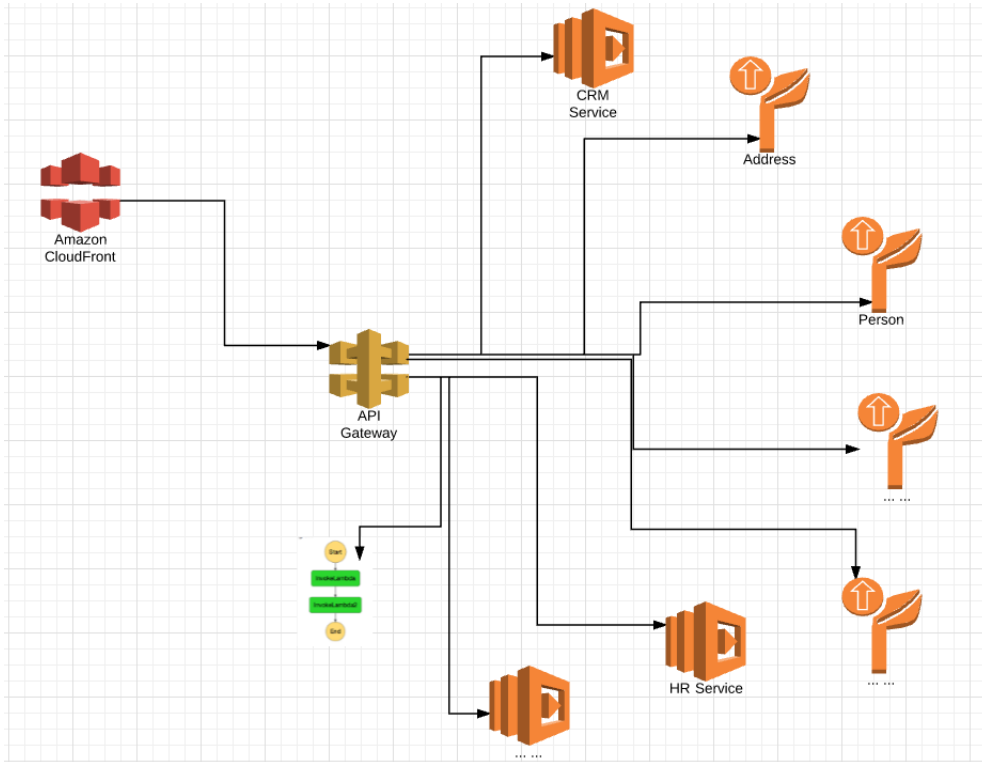
## An API call flow



# Authorization



# Target Architecture



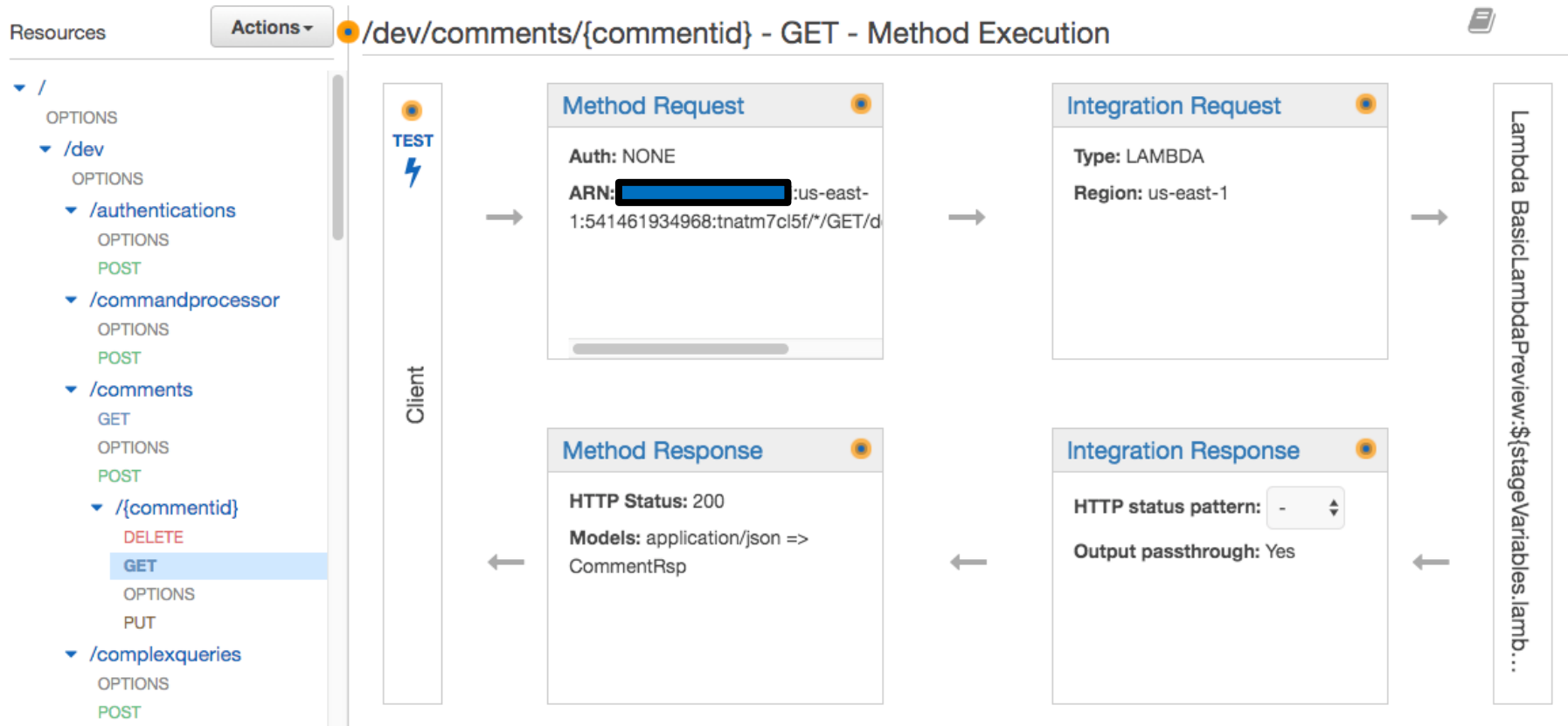
- Microservices will interact “directly” (not shown) via
  - REST
  - SNS
  - SQS
- The Lambda functions will
  - Implement composition microservices in code.
  - Simulate “complex” applications.
- We will add Step Functions to
  - Implement some complex processes
  - Understand orchestration engine relative to code implementation.
- API Gateway
  - Implement consistent API.
  - API Management functions.
  - Implement authentication and authorization.
- CloudFront provides single
  - Site/domain image.
  - SSL termination/certificates.

# API Gateway/Management

The screenshot displays the Amazon API Gateway console interface. At the top, there is a header bar with the Amazon API Gateway logo and the text "APIs". Below the header, a sidebar on the left lists various APIs: AssetManager, CacheTest, Crowdfunding Preview, Equals3Catalog, GetUrl, LambdaMicroservice, MailService, MultiTenantCatalog, MultiTenantCustomerInfo, MultiTenantQuery, newcatalogapi, pageinfo, Placement Service, Seeka TV Catalog API, Seeka TV Catalog API -- Fran..., and Seeka TV Catalog API -- Series. The main content area features a "Create API" button at the top left. Below this button, a grid of API cards is displayed, each with a title and a description. The APIs shown in the grid are: AssetManager (No description), CacheTest (No description), Crowdfunding Preview (Crowdfunding Preview), Equals3Catalog (New equals3 catalog going to Neo4J and proxy model), GetUrl (Created by AWS Lambda), LambdaMicroservice (Created by AWS Lambda), MailService (Facade for wrapping MailChimp and Mandrill), MultiTenantCatalog (Multitenant catalog), MultiTenantCustomerInfo (Multitenant JavaScript CustomerInfo), MultiTenantQuery (Multitenant JavaScript Query Service), newcatalogapi (New Catalog API), and pageinfo (No description).

API Name	Description
AssetManager	No description.
CacheTest	No description.
Crowdfunding Preview	Crowdfunding Preview
Equals3Catalog	New equals3 catalog going to Neo4J and proxy model
GetUrl	Created by AWS Lambda
LambdaMicroservice	Created by AWS Lambda
MailService	Facade for wrapping MailChimp and Mandrill.
MultiTenantCatalog	Multitenant catalog
MultiTenantCustomerInfo	Multitenant JavaScript CustomerInfo
MultiTenantQuery	Multitenant JavaScript Query Service
newcatalogapi	New Catalog API
pageinfo	No description.

# Walkthrough of API



# Method

Resources

Actions ▾

/


/CRMService

ANY

/hrservice


GET


/hrservice - GET - Setup





Choose the integration point for your new method.

Integration type

☐ Lambda Function 

☐ HTTP 

☐ Mock 

☒ AWS Service 

AWS Region

AWS Service

AWS Subdomain

HTTP method


Action Type

☒ Use action name


☐ Use path override

Action

Execution role



Content Handling



Save



# Walkthrough of API

Resources Actions ▾ /dev/comments/{commentid} - GET - Method Execution

TEST

Method Request

Auth: NONE

Integration Request

Type: LAMBDA

**Demo**

Lambda BasicLambdaPreview:\${stageVariables.lamb...

CommentRsp

GET

OPTIONS

PUT

POST

# Step Functions

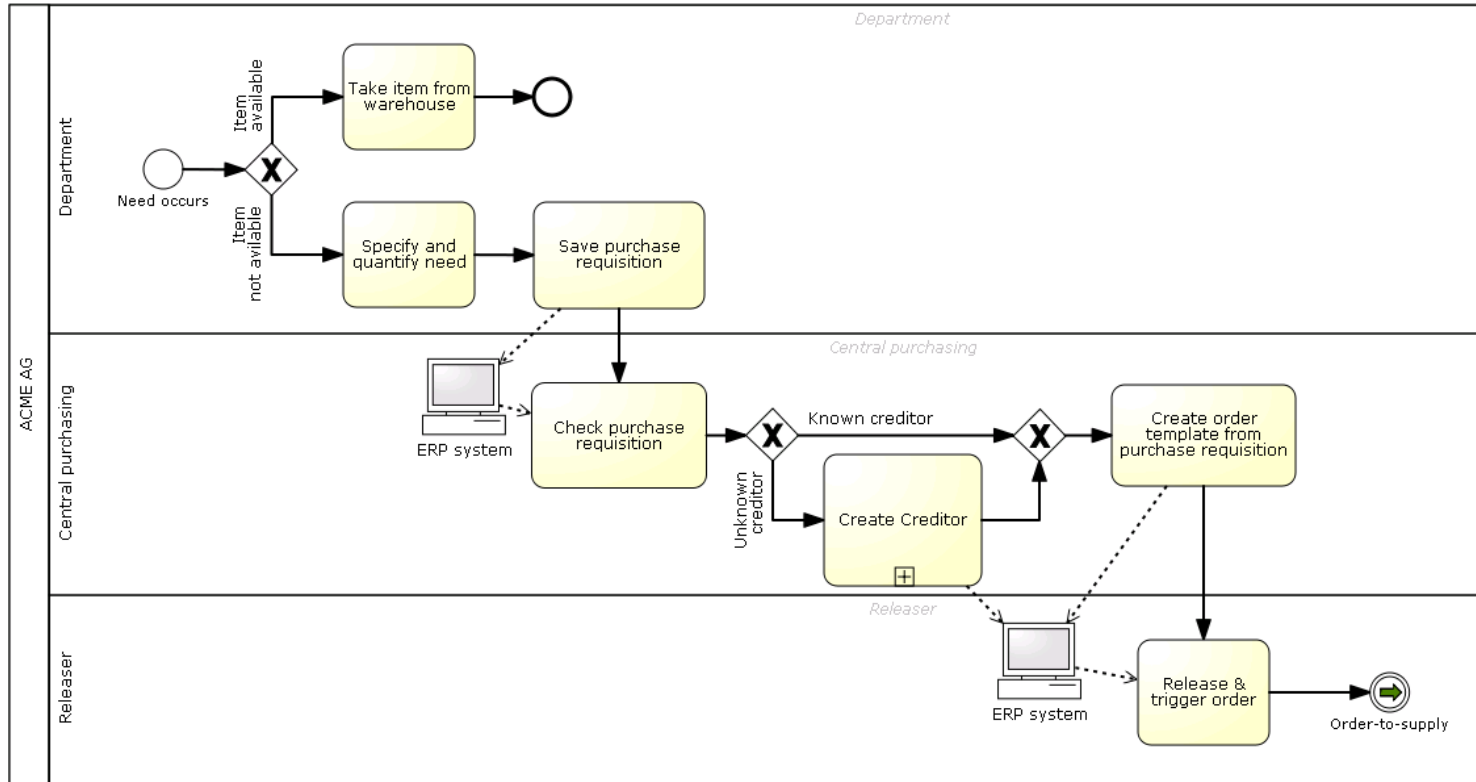
# Workflow Orchestration

# Workflow/Orchestration

“A **workflow** consists of an orchestrated and repeatable pattern of business activity enabled by the systematic organization of resources into processes that transform materials, provide services, or process information.<sup>[1]</sup> It can be depicted as a sequence of operations, declared as work of a person or group,<sup>[2]</sup> an organization of staff, or one or more simple or complex mechanisms.” (<https://en.wikipedia.org/wiki/Workflow>)

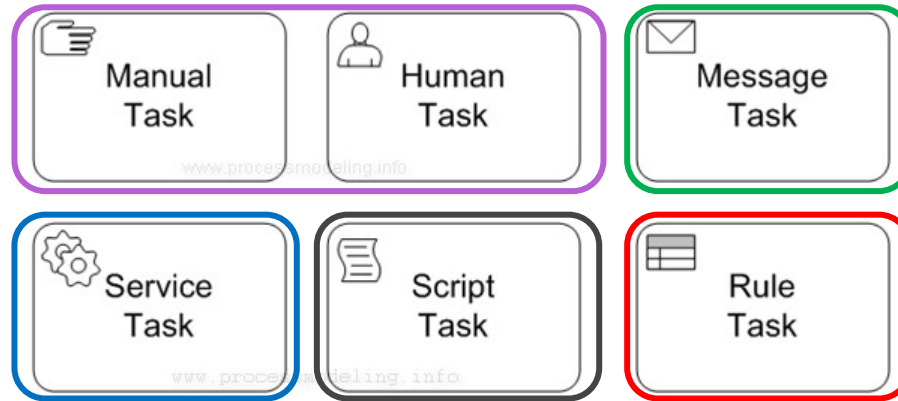
“Cloud computing introduces more-granular and specific meanings of the terms "workflows" and "processes" as used in different domains. Processes can be identified at both the level of inter-company business and the level of wide area network ("the cloud") operations. An "orchestrator" is understood to be the entity which manages complex cross-domain (system, enterprise, firewall) processes and handles exceptions. Since an orchestrator is valuable in fulfillment, assurance, and billing processes,<sup>[4]</sup> service-aware incarnations of an orchestrator should be capable of adjustments based on feedback from monitoring tools. At the most basic level, an orchestrator is a human.” ([https://en.wikipedia.org/wiki/Orchestration\\_\(computing\)](https://en.wikipedia.org/wiki/Orchestration_(computing)))

# Simple BPMN Diagram



# BPMN – How do you Implement?

- Notify a person/role.
- Have them do something.
- Have the use an app that calls the engine to tell you they are done.



- Send an email.
- POST a document
- etc.

- Decision table
- Call Drools
- etc.

- Insert into a DB
- Call a web service
- Run a Java app.
- etc.
- Run some JavaScript or Java or ...
- Right at this point in the workflow/process
- And manipulate the data/documents/control.

# Modeling/Pictures (an aside)

<https://www.nomagic.com/getting-started/modeling-benefits>

## Modeling Benefits

### Manage Complexity

Modeling is essential in complexity management. Modeling benefits include:

- Viewing systems from multiple perspectives
- Discovering causes and effects using model traceability
- Improving system understanding through visual analysis
- Discovering errors earlier and reducing system defects
- Exploring alternatives earlier in the system lifecycle
- Improving impact analysis, identifying potential consequences of a change, or estimating modifications to implement a change
- Simulating system solutions without code generation

# Modeling/Pictures (an aside)

<https://www.nomagic.com/getting-started/modeling-benefits>

## Preserve Knowledge and Corporate Memory

Modeling helps enterprises preserve knowledge and corporate memory by:

- Storing the corporate memory in a versioned repository
- Enabling quick and easy understanding of your systems within an organization by all members of your teams
- Assisting new team members in getting up to speed quickly

## Reuse

Modeling helps you reuse parts of existing information and knowledge in your new projects, saving time and money.

## Automate

Modeling facilitates automation including these examples:

- Automate generation of a real working system or part of a system from models
- Automate repeatable tasks by writing scripts
- Use thousands of shortcuts and features for getting the expected results in a single click



# Step Functions (Introduction)

# AWS Step Functions

(<https://aws.amazon.com/step-functions/>)



## Productivity:

### Build Applications Quickly

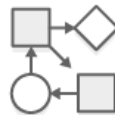
AWS Step Functions includes a visual console and blueprints for commonly-used workflows that make it easy to coordinate the components of distributed applications into parallel and/or sequential steps. You can build applications in a matter of minutes, and then visualize and track the execution of each step to help ensure the application is operating as intended.



## Resilience:

### Scale and Recover Reliably

AWS Step Functions automatically triggers each step so your application executes in order and as expected. It can handle millions of steps simultaneously to help ensure your application is available as demand increases. Step Functions tracks the state of each step and handles errors with built-in retry and fallback, whether the step takes seconds or months to complete.



## Agility:

### Evolve Applications Easily

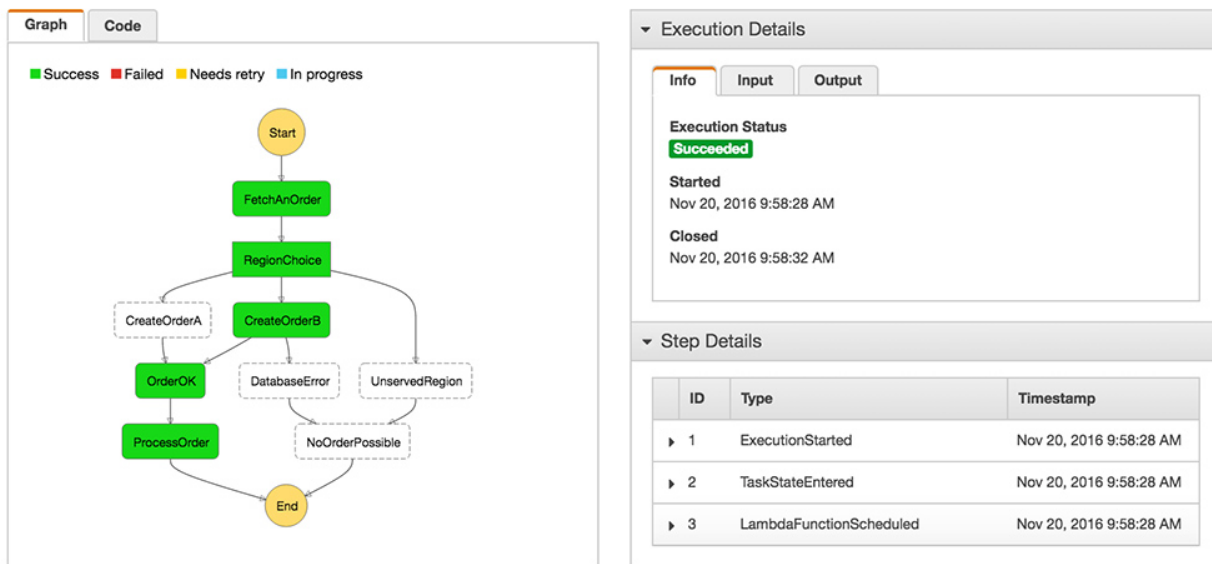
AWS Step Functions makes it easy to change workflows and edit the sequence of steps without revising the entire application. You can re-use components and steps without even changing their code to experiment and innovate faster. Your workflow can support thousands of individual components and steps, so you can freely build increasingly complex applications.

# Execute and Monitor

## 2. Verify Everything is Operating as Intended

Start an execution to visualize and verify the steps of your application are operating as intended. The console highlights the real-time status of each step and provides a detailed history of every execution.

### AWS Step Functions



# Simple Example

GraphCode

■ Success ■ Failed ■ Cancelled ■ In progress

```
graph TD; Start((Start)) --> InvokeLambda[InvokeLambda]; InvokeLambda --> InvokeLambda2[InvokeLambda2]; InvokeLambda2 --> End((End));
```

Execution Details

InfoInputOutput

Execution Status

Succeeded

State Machine Arn

arn:aws:states:us-east-1:832720255830:stateMachine:secondmachine

Execution ID

arn:aws:states:us-east-1:832720255830:execution:secondmachine:dc977240-d68a-86ca-e4eb-375648105405

Started

Jan 28, 2017 11:00:36 AM

Closed

Jan 28, 2017 11:00:37 AM

Step Details

ID	Type	Timestamp
▶ 1	ExecutionStarted	Jan 28, 2017 11:00:36 AM
▶ 2	TaskStateEntered	Jan 28, 2017 11:00:36 AM
▶ 3	LambdaFunctionScheduled	Jan 28, 2017 11:00:36 AM
▶ 4	LambdaFunctionStarted	Jan 28, 2017 11:00:36 AM
▶ 5	LambdaFunctionSucceeded	Jan 28, 2017 11:00:36 AM
▶ 6	TaskStateExited	Jan 28, 2017 11:00:36 AM
▶ 7	TaskStateEntered	Jan 28, 2017 11:00:36 AM

# Simple Example

Graph Code

```
1 {  
2   "Comment": "A Hello World example of the Amazon States Language using an AWS Lambda  
3   Function",  
4   "StartAt": "InvokeLambda",  
5   "States": {  
6     "InvokeLambda": {  
7       "Type": "Task",  
8       "Resource": "arn:aws:lambda:us-east-1:832720255830:function:helloworld",  
9       "Next": "InvokeLambda2"  
10    },  
11    "InvokeLambda2": {  
12      "Type": "Task",  
13      "Resource": "arn:aws:lambda:us-east-1:832720255830:function:columbiaecho",  
14      "End": true  
15    }  
16  }  
17 }
```

# Concepts

## ■ AWS Step Functions Concepts

- Amazon States Language
- States
- Tasks
- Transitions
- State Machine Data
- Executions
- Error Handling
- Creating IAM Roles for Use with AWS Step Functions

```
{
  "Comment": "An Amazon States Language example using a Choice state.",
  "StartAt": "FirstState",
  "States": {
    "FirstState": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:FUNCTION_NAME",
      "Next": "ChoiceState"
    },
    "ChoiceState": {
      "Type": "Choice",
      "Choices": [
        {
          "Variable": "$.foo",
          "NumericEquals": 1,
          "Next": "FirstMatchState"
        },
        {
          "Variable": "$.foo",
          "NumericEquals": 2,
          "Next": "SecondMatchState"
        }
      ],
      "Default": "DefaultState"
    },
    "FirstMatchState": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:OnFirstMatch",
      "Next": "NextState"
    },
    "SecondMatchState": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:OnSecondMatch",
      "Next": "NextState"
    }
  }
}
```

# Tasks and Activities

## Tasks

All work in your state machine is done by *tasks*. A task can be:

- An Activity, which can consist of any code in any language. Activities can be hosted on EC2, ECS, mobile devices—basically anywhere. Activities must poll AWS Step Functions using the `GetActivityTask` and `SendTask*` API calls. (Ultimately, an activity can even be a human task—a task that waits for a human to perform some action and then continues.)
- A **Lambda function**, which is a completely cloud-based task that runs on the **Lambda** service. Lambda functions can be written in JavaScript (which you can write using the AWS Management Console or upload to Lambda), or in Java or Python (uploaded to Lambda).

Tasks are represented in Amazon States Language by setting a state's type to `Task` and providing it with the ARN of the created activity or Lambda function. For details about how to specify different task types, see **Task** in the [Amazon States Language Overview](#).

# OAuth



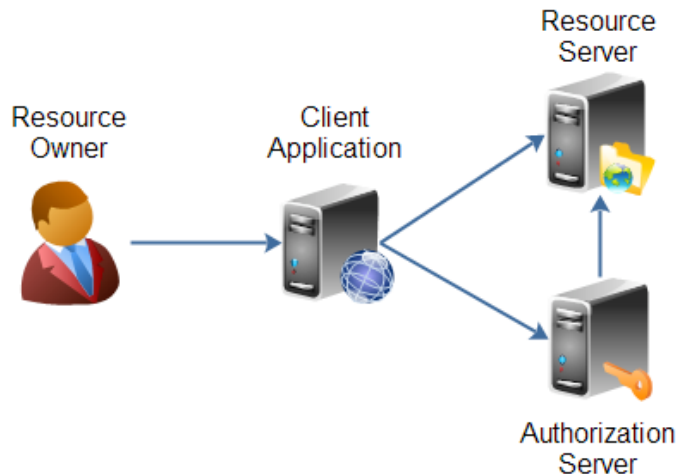
# Overview (<http://tutorials.jenkov.com/oauth2/index.html>)

- Resource Owner
  - Controls *access* to the “data.”
  - Facebook **user**, LinkedIn **user**, ...
- Resource Server
  - The website that holds/manages info.
  - Facebook, LinkedIn, ...
  - And provides access API.
- Client Application
  - “The product you implemented.”
  - Wants to read/update
    - “On your behalf”
    - The data the data that the Resource Server maintains, e.g. posts, status, tweets, ...
- Authorization Server
  - Grants/rejects authorization
  - Based on Resource Owner decisions.
  - Usually (logically) the same as Resource Server.

OAuth 2.0 defines the following roles of users and applications:

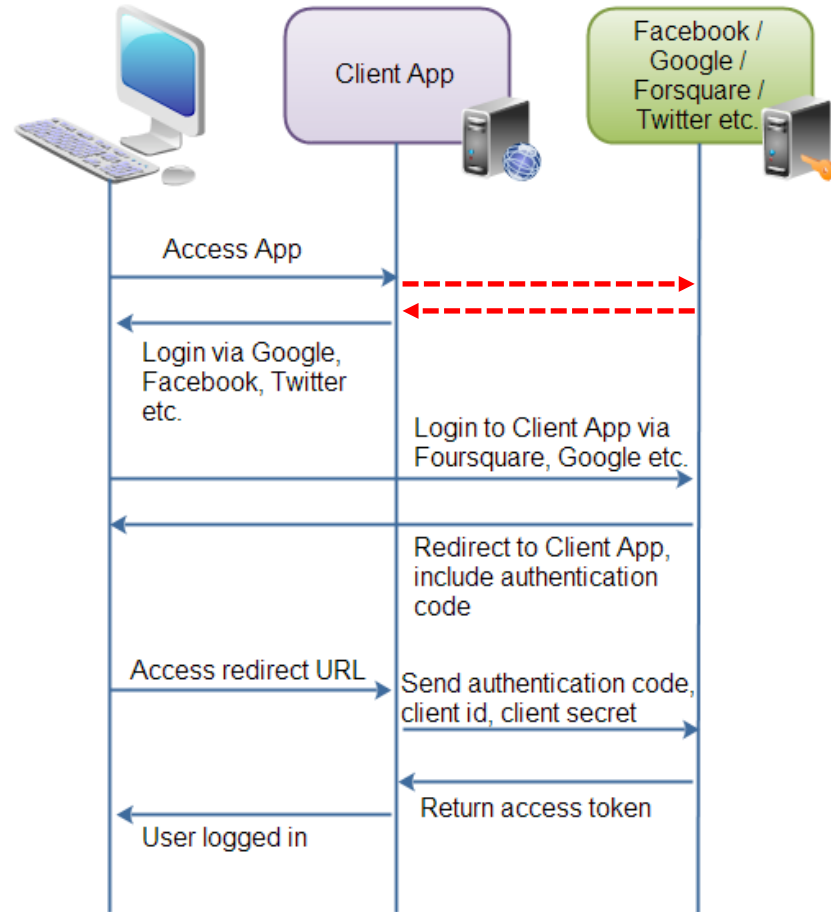
- Resource Owner
- Resource Server
- Client Application
- Authorization Server

These roles are illustrated in this diagram:



# Roles/Flows

- User Clicks “Ligon with XXX”
  - Redirect user to XXX
  - With Client App application ID.
  - And permissions app requests.
  - Code MAY have to call Resource Server to get a token to include in redirect URL.
- Browser redirected to XXX
  - Logon on to XXX prompt.
  - Followed by “do you grant permission to ...”
  - Clicking button drives a redirect back to Client App. URL contains a temporary token.
- User/Browser
  - Redirected to Client App URL with token.
  - Client App calls XXX API
  - Obtains access token.
  - Returns to User.
- Client App can use access token on API calls.



# Demo

- Facebook application
- Test Users
- Login/Registration flow
- Graph Explorer