COMS E6998: **Microservices and Cloud Applications**

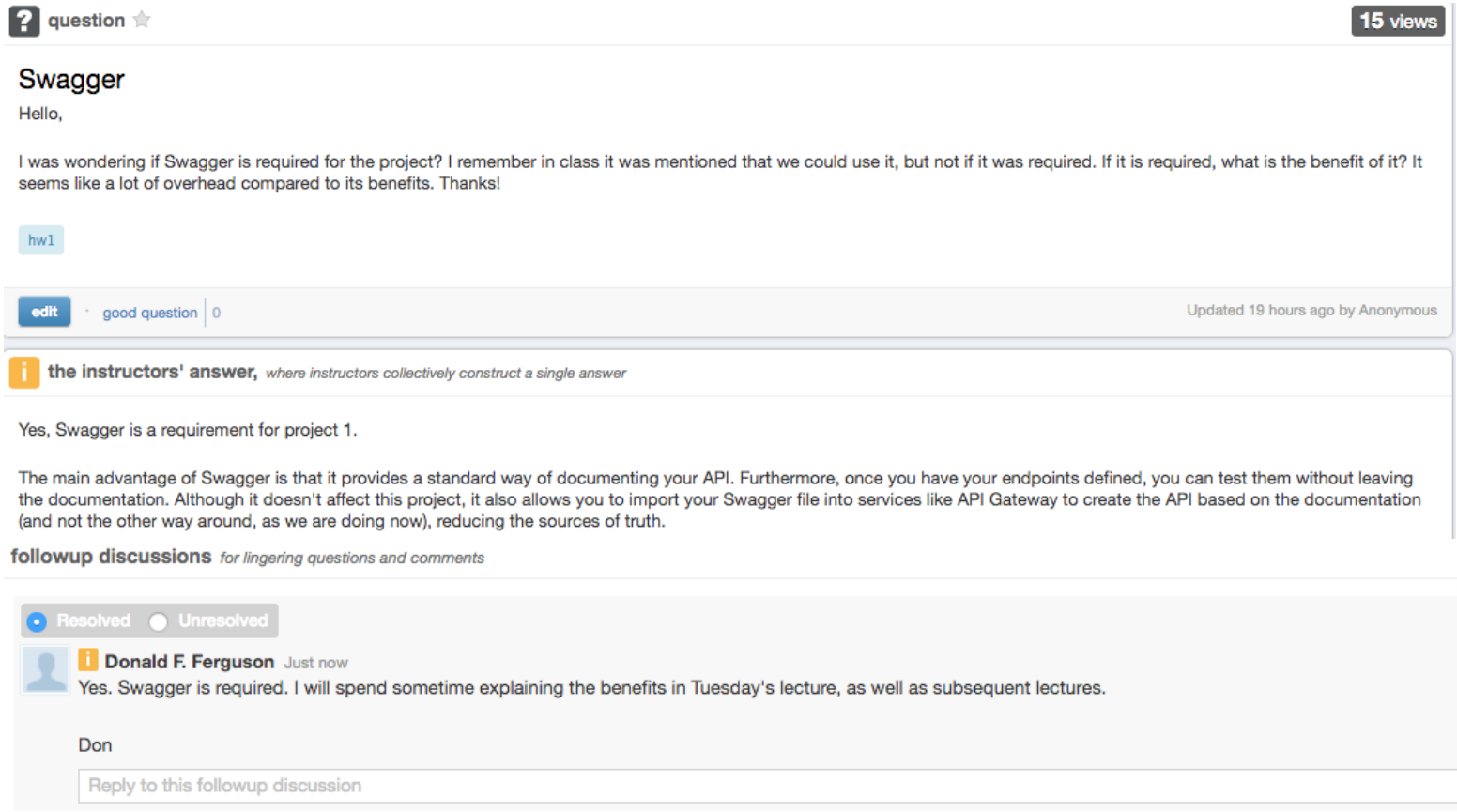
Lecture 4: Composition, Integration, Orchestration, Pub/Sub, Message Queueing

Dr. Donald F. Ferguson
dff9@columbia.edu

Questions? Comments?

Swagger

Swagger



question

15 views

Swagger

Hello,

I was wondering if Swagger is required for the project? I remember in class it was mentioned that we could use it, but not if it was required. If it is required, what is the benefit of it? It seems like a lot of overhead compared to its benefits. Thanks!

hw1

edit · good question | 0

Updated 19 hours ago by Anonymous

the instructors' answer, *where instructors collectively construct a single answer*

Yes, Swagger is a requirement for project 1.

The main advantage of Swagger is that it provides a standard way of documenting your API. Furthermore, once you have your endpoints defined, you can test them without leaving the documentation. Although it doesn't affect this project, it also allows you to import your Swagger file into services like API Gateway to create the API based on the documentation (and not the other way around, as we are doing now), reducing the sources of truth.

followup discussions *for lingering questions and comments*

Resolved Unresolved

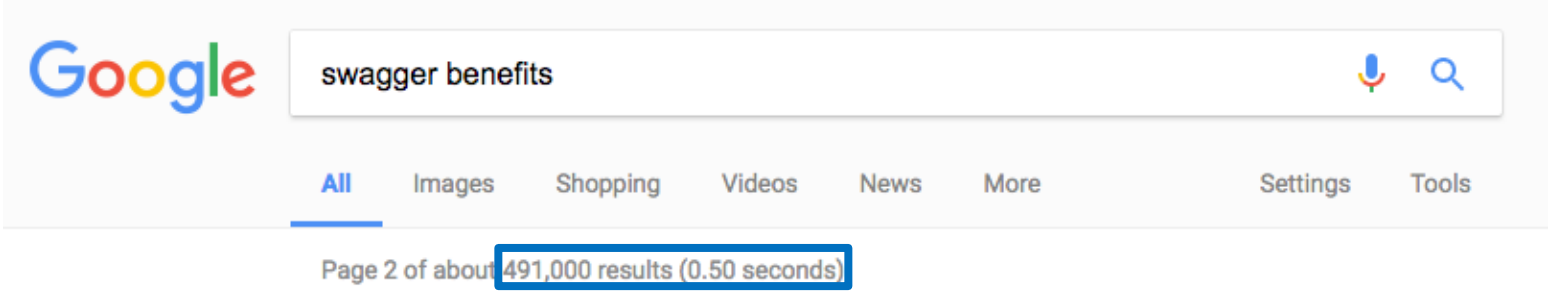
Donald F. Ferguson Just now

Yes. Swagger is required. I will spend sometime explaining the benefits in Tuesday's lecture, as well as subsequent lectures.

Don

Reply to this followup discussion

Swagger Benefits

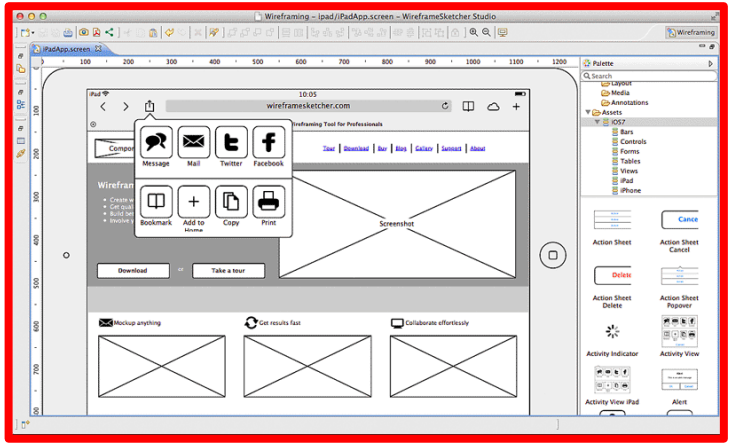


OK. No search is perfect.

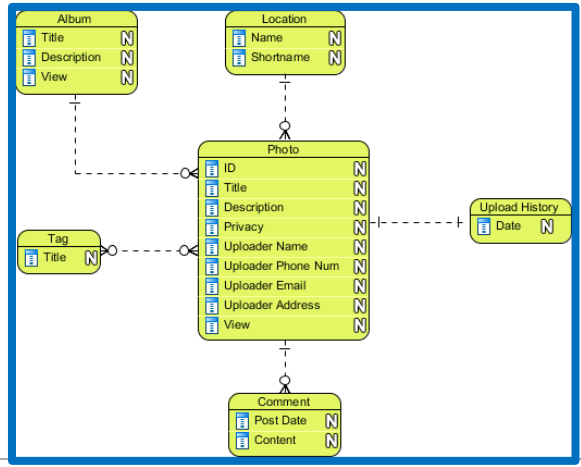
The global benefits of Trump's swagger - The Week
theweek.com/articles/670750/global-benefits-trumps-swagger ▾
Jan 9, 2017 - Barack Obama flunked on foreign affairs. And believe it or not, Donald Trump can do better. Some Obama failures (Iraq, Afghanistan) have ...

Reminder: (Some) Application Design Methodologies

- Start with User Interface
  - Roles and personas.
  - Wireframes for pages and interactions.
  - Page flow/transition diagrams.

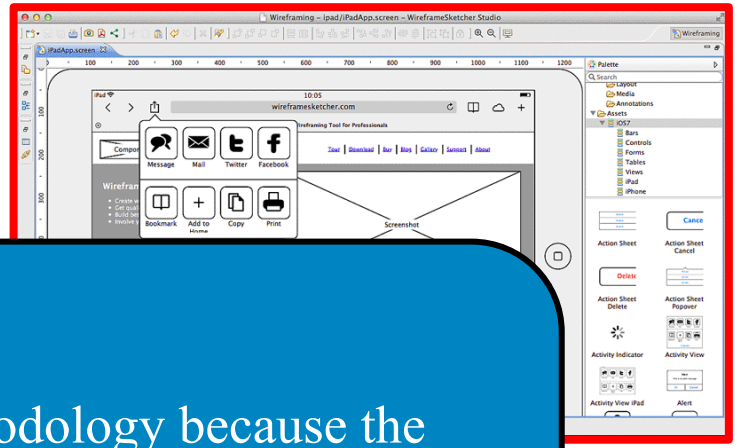


- Start with Data Model
  - Entity types, properties.
  - Relationships/associations.
  - Constraints.
  - Operations.



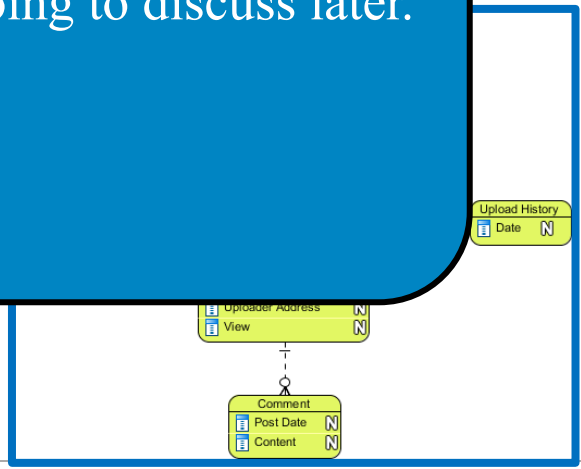
Reminder: (Some) Application Design Methodologies

- Start v
  - Ro
  - Wi
  - Pa
- Start
  - En
  - Rel
  - Constraints.
  - Operations.



I intentionally omitted a methodology because the concepts is less familiar and we are going to discuss later.

API First.



API Economy (http://www.gartner.com/smarterwithgartner/welcome-to-the-api-economy/)

Smarter With Gartner

The API economy is an enabler for turning a business or organization into a platform.

We live in an API economy, a set of business models and channels based on secure access of functionality and exchange of data. APIs make it easier to integrate and connect people, places, systems, data, things and algorithms, create new user experiences, share data and information, authenticate people and things, enable transactions and algorithms, leverage third-party algorithms, and create new product/services and business models.

"The API economy is an enabler for turning a business or organization into a platform." said Kristin R. Moyer, vice president and distinguished analyst at Gartner. "Platforms multiply value creation because they enable business ecosystems inside and outside of the enterprise to consummate matches among users and facilitate the creation and/or exchange of goods, services and social currency so that all participants are able to capture value."

Uber, for instance, is an example of a business built on a platform because it leverages Google Maps through an API to enable its entire business model of matching drivers who have a vehicle with passengers who need a ride. Walgreens offers an API for its in-store photo printing services that enables others to offer photo apps on its platform. It moves from being a photo printer to being a photo platform.

2017 Is Quickly Becoming The Year Of The API Economy

https://www.forbes.com/sites/louiscolumbus/2017/01/29/2017-is-quickly-becoming-the-year-of-the-api-economy/#4b3806fe6a41

Louis Columbus, CONTRIBUTOR
FULL BIO
Opinions expressed by Forbes Contributors are their own.

This year more CIOs will have their bonuses tied to how many new business models they help create with existing and planned IT platforms than ever before. This trend will accelerate over the next three years. CIOs and IT staffs need to start thinking about how they can become business strategists first, technicians and enablers of IT second. CIOs must create and launch new business models faster to keep their companies competitive. APIs are the fuel helping to make this happen.

The Urgency To Create New Business Models Is Driving API Proliferation

APIs (Application Programmer Interfaces) are the components that enable diverse platforms, apps, and systems to connect and share data with each other. Think of APIs as a set of software modules, tools, and protocols that enable two or more platforms, systems and most commonly, applications to communicate with each other and initiate tasks or processes. APIs are essential for defining and customizing Graphical User Interfaces (GUIs) too. Cloud platform providers all have extensive APIs defined and work in close collaboration with development partners to fine-tune app performance using them. Amazon Web Services, Facebook, Google, Marketo, Salesforce, SAP Hybris, Twitter and thousands of other companies have APIs available. As of today, the Programmable Web lists 16,590 APIs in its database.

A Perspective (https://blog.readme.io/what-is-swagger-and-why-it-matters/)

In the software industry, the API landscape is facing a similar issue. Great code is crucial for modern businesses, and the best way for us to connect and share data is through APIs. But not only has there been no industry standard for designing APIs, there hasn't been an industry standard for documenting them.

APIs are supposed to connect engineers and allow for the sharing of great developments. APIs let companies like Twilio add value to other products and create an ecosystem of shared knowledge. But an API is only valuable if it's accessible. And for that, it needs clear documentation.

Terrible documentation is just as useless as a clock that tells the wrong time. So developers have worked hard to find a way to standardize the vocabulary surrounding APIs. That's where Swagger comes in. Swagger is basically a set of rules (specification) and tooling for how to semantically describe APIs. Practically, it's a language-agnostic tool that gets everyone on the same page.

A Perspective (https://blog.readme.io/what-is-swagger-and-why-it-matters/)

There are other available frameworks that have gained some popularity, such as RAML, APIBlueprint, and Summation. But Swagger provides more benefits than just helping create clear documentation.

- **It's comprehensible for developers and non-developers.** Product managers, partners, and even potential clients can have input into the design of your API, because they can see it clearly mapped out in this friendly UI.
- **It's human readable *and* machine readable.** This means that not only can this be shared with your team internally, but the same documentation can be used to automate API-dependent processes.
- **It's easily adjustable**. This makes it great for testing and debugging API problems.

These three benefits not only make developers' lives easier, but they make the API more consumable. Any API that adheres to the Swagger spec is easy to read, easy to iterate, and easy to consume. That's why huge companies like Netflix, Akana, and Yelp have already jumped on the Swagger train.

A Perspective (https://blog.readme.io/what-is-swagger-and-why-it-matters/)

The rise of design-first API

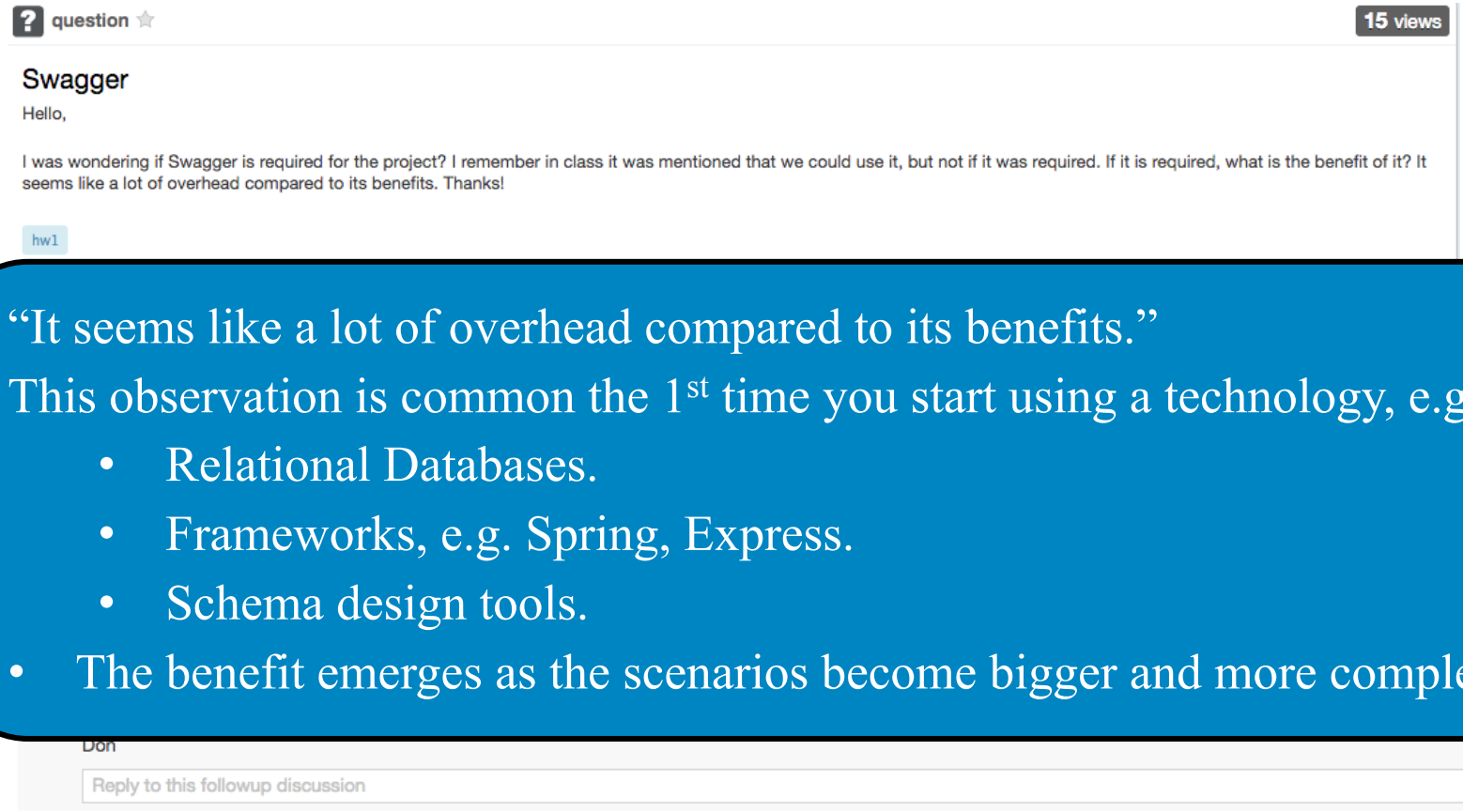
With this blueprint in mind, there are two main ways to take advantage of Swagger:

- **Top-down approach, or design-first.** This means you're using Swagger to design your API before you've written any actual code.
- **Bottom-up approach, or code-first.** This means you've already written the code for your API, and you'll be using Swagger to document your API.

In the early days, it was popular for APIs to be created code-first. This is much easier because you can make adjustments as you go, and it fits nicely into an Agile delivery process. But because you're not thinking about the design, this can make for an API that's difficult to understand and document.

The push for clear, easy-to-read documentation has popularized the design-first approach. Not only can more people have input on your documentation, but it actually results in cleaner code. You're forced to think simpler, more concise, and easy-to-follow.

Swagger



"It seems like a lot of overhead compared to its benefits."

This observation is common the 1st time you start using a technology, e.g.

- Relational Databases.
- Frameworks, e.g. Spring, Express.
- Schema design tools.

- The benefit emerges as the scenarios become bigger and more complex.

Complex Application Environments

CRM

In CRM (customer relationship management), CRM software is a category of software that covers a broad set of applications designed to help businesses manage many of the following business processes:

- customer data
- customer interaction
- access business information
- automate sales
- track leads
- contracts
- marketing
- customer support
- clients and contacts
- support vendor / partner relationships
- employees
- knowledge and training
- assets or resources

While CRM software is most commonly used to manage a business-customer relationship, CRM software systems are also used in the same way to manage business contacts, employees, clients, contract wins and sales leads. Typically, CRM software is used in the enterprise, however many products scale and can be used in a business of any size.

http://www.webopedia.com/TERM/C/crm_software.html

http://solutionleadership.blogspot.com/2010/02/how-to-lead-crm-solution.html

Enterprise Resource Planning

Enterprise resource planning (**ERP**) is the integrated management of core business processes, often in real-time and mediated by software and technology. These business activities can include:

- product planning, purchase
- production planning
- manufacturing or service delivery
- marketing and sales
- materials management
- inventory management
- shipping and payment
- finance

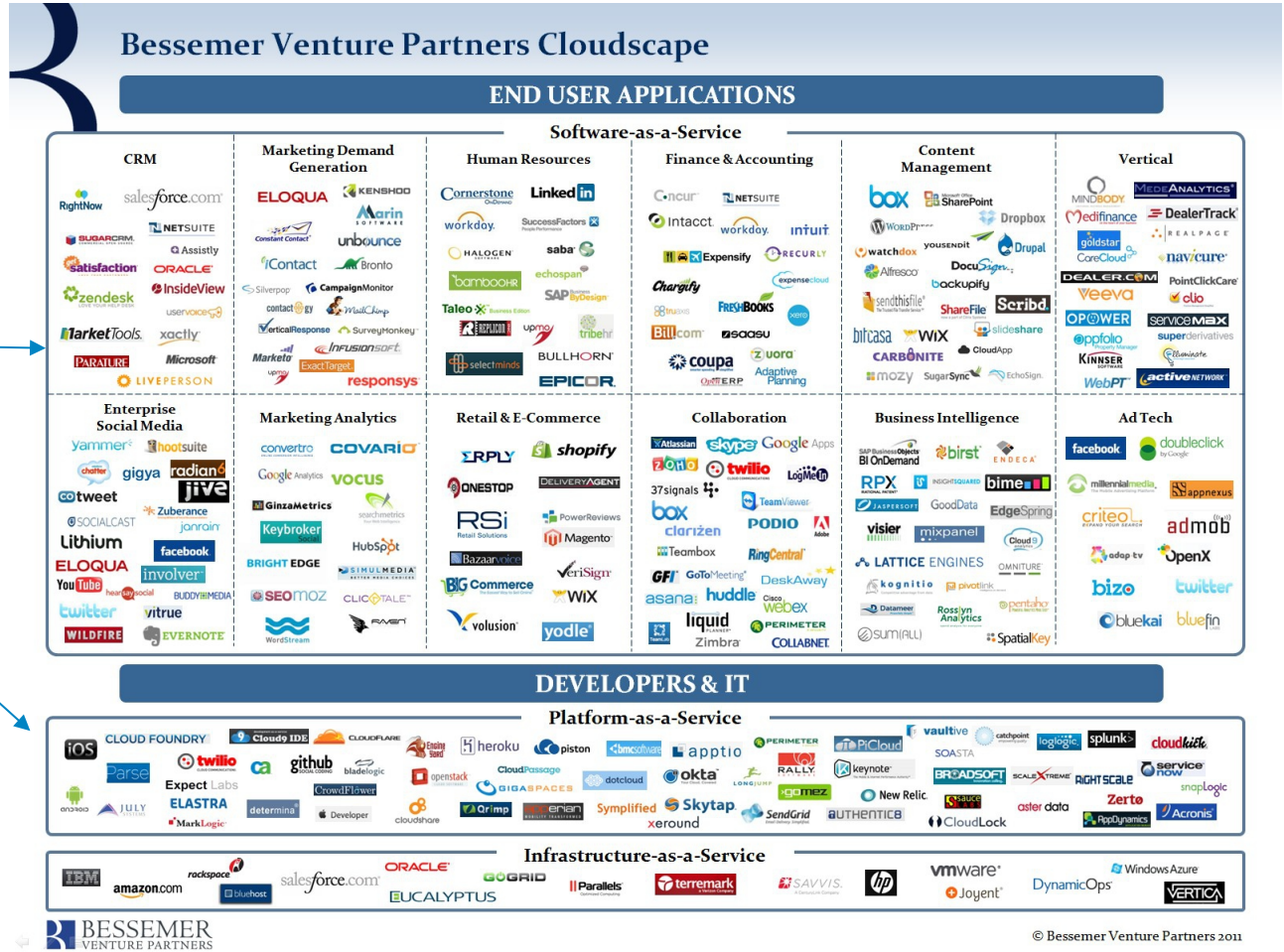
ERP is usually referred to as **category** of business-management software—typically a suite of integrated applications—that an organization can use to collect, store, manage and interpret data from these many business activities.

I Can Do This All Day

Cloudscape

Our applications will

- Call APIs
- Use and run in PaaS



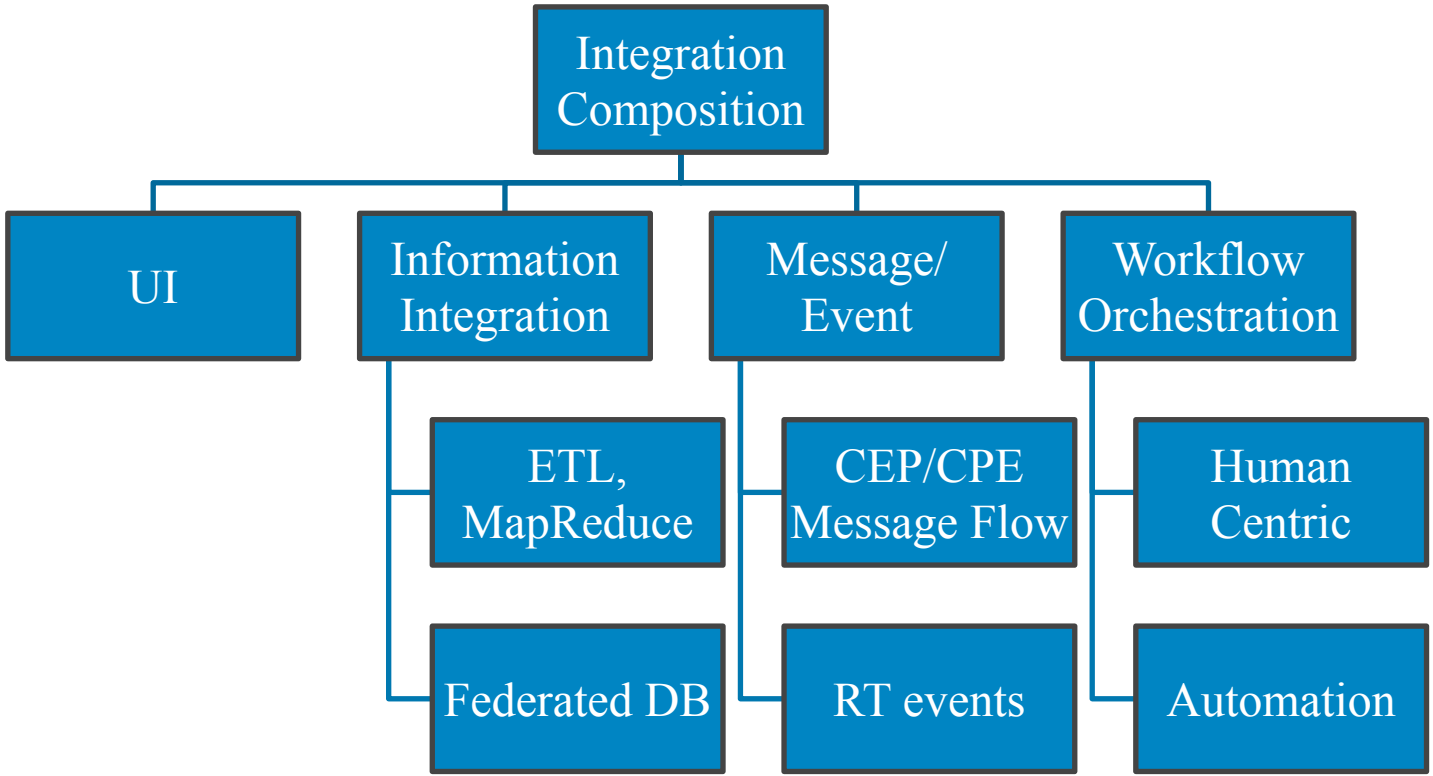
That was interesting, but why did you present?

- These are some enterprise software examples, but the pattern is general.

- There are three broad classifications of microservices
  1. Basic building blocks for core functions.
  2. Microservices that *integrate* and *compose* building blocks into solutions.
  3. Microservices that *integrate* and *compose* solutions across the enterprise, and with other enterprises and partners.

- This leads to three fundamental considerations.
  1. Making individual microservices *composable*.
  2. *Design patterns* and *technology* for integration and composition.
  3. A large part of microservice and cloud development is composing, integrating and extending SaaS delivered solutions via APIs.

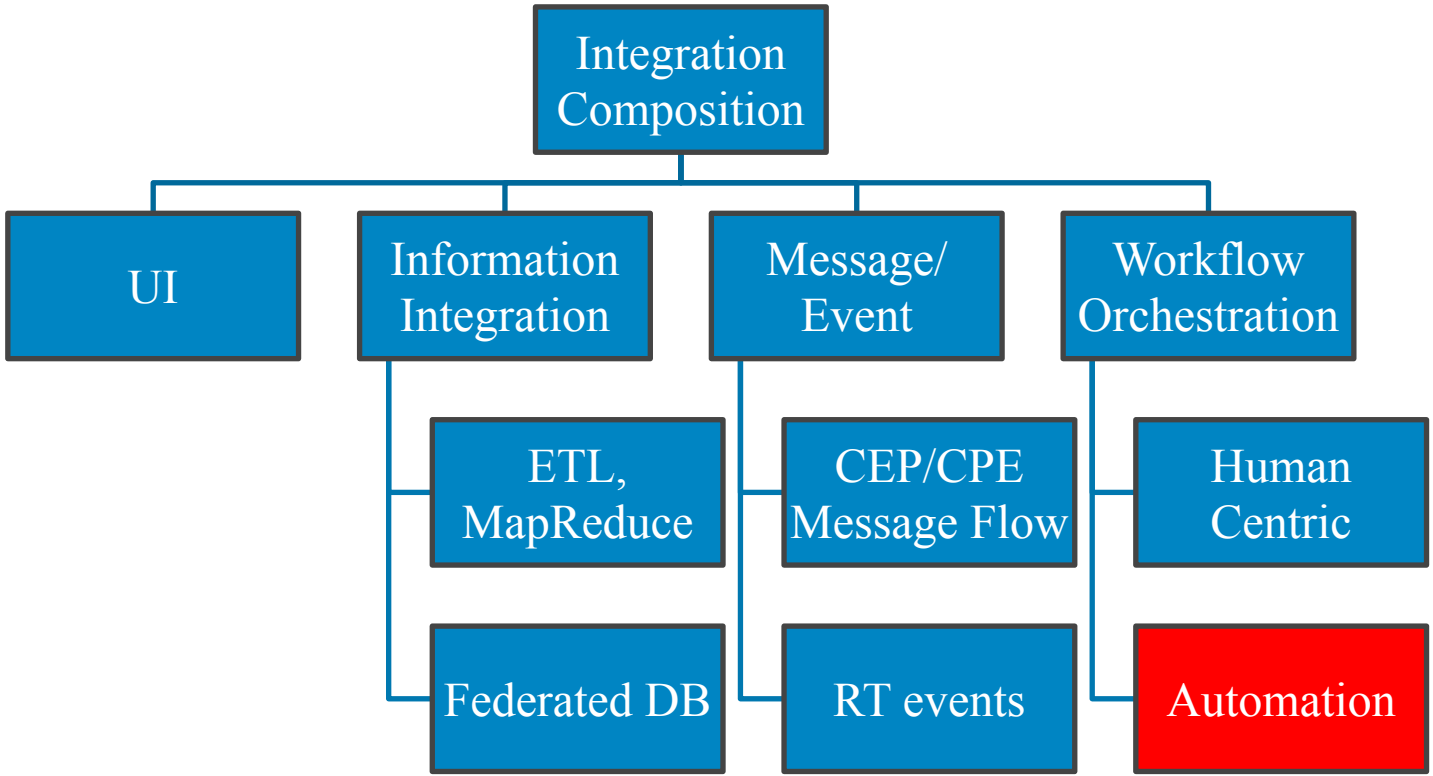
Simplistic Taxonomy of Integration/Composition



Automation (Workflow) (Orchestration)

Introduction

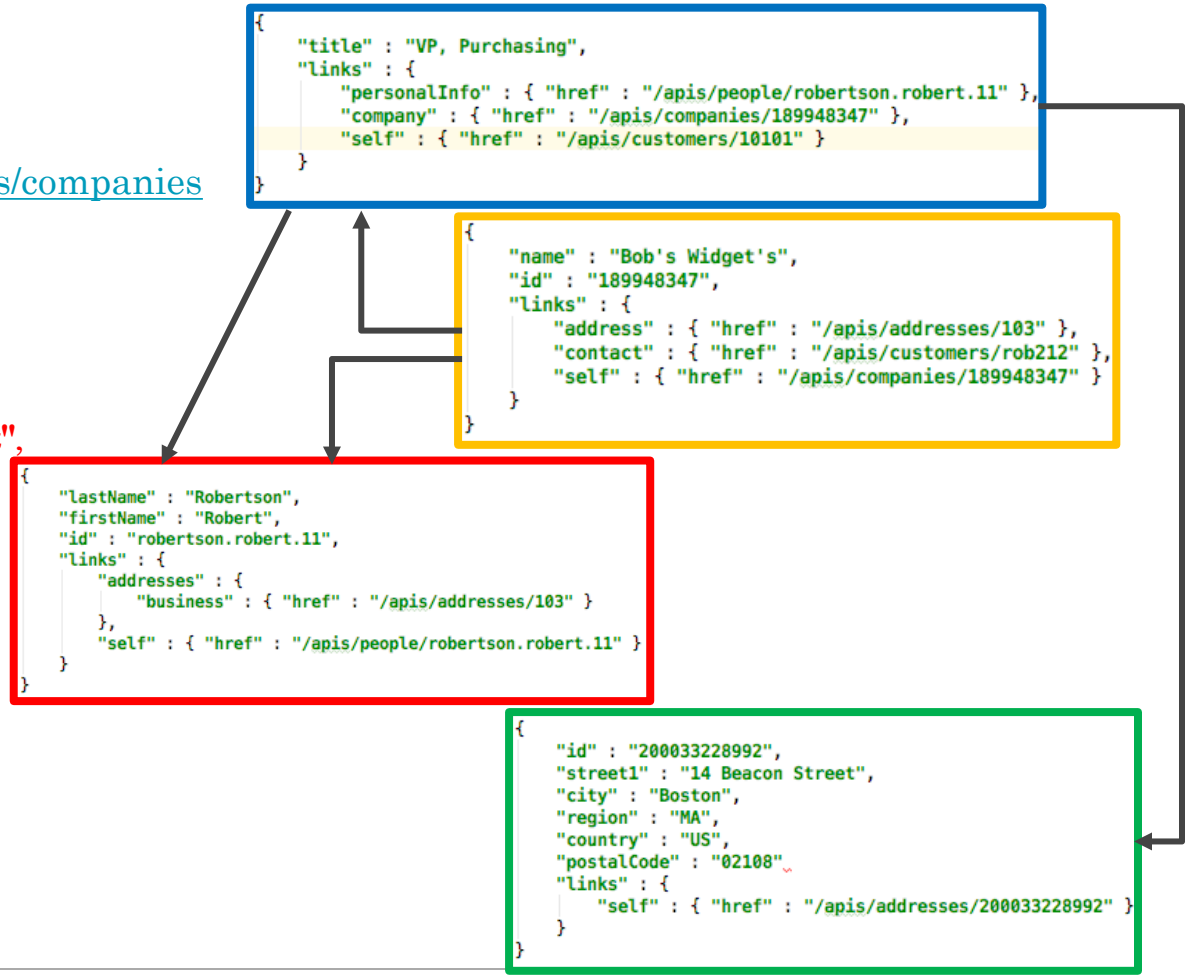
Simplistic Taxonomy of Integration/Composition



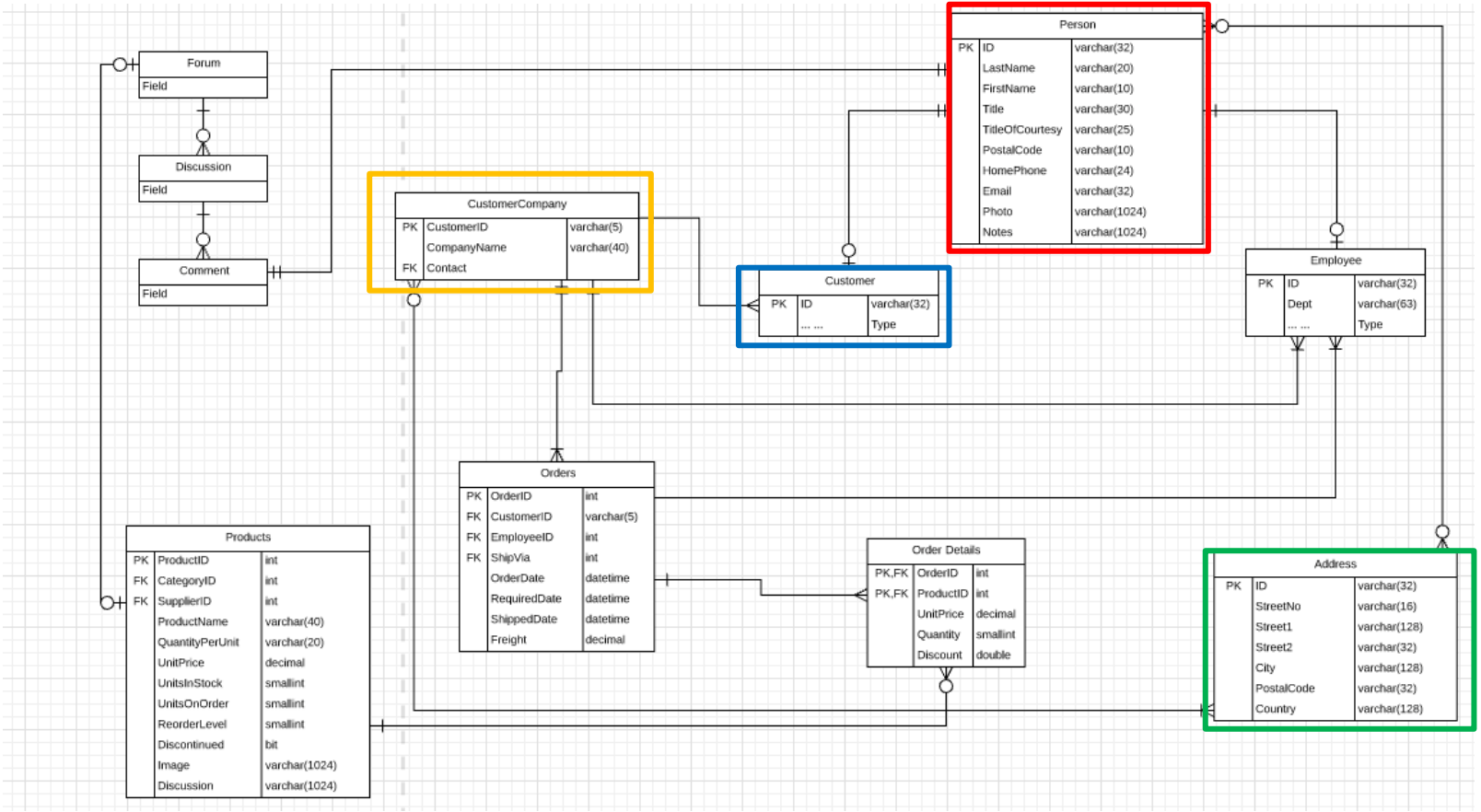
CreateCustomer

- POST HTTPS://contoso.com/apis/companies

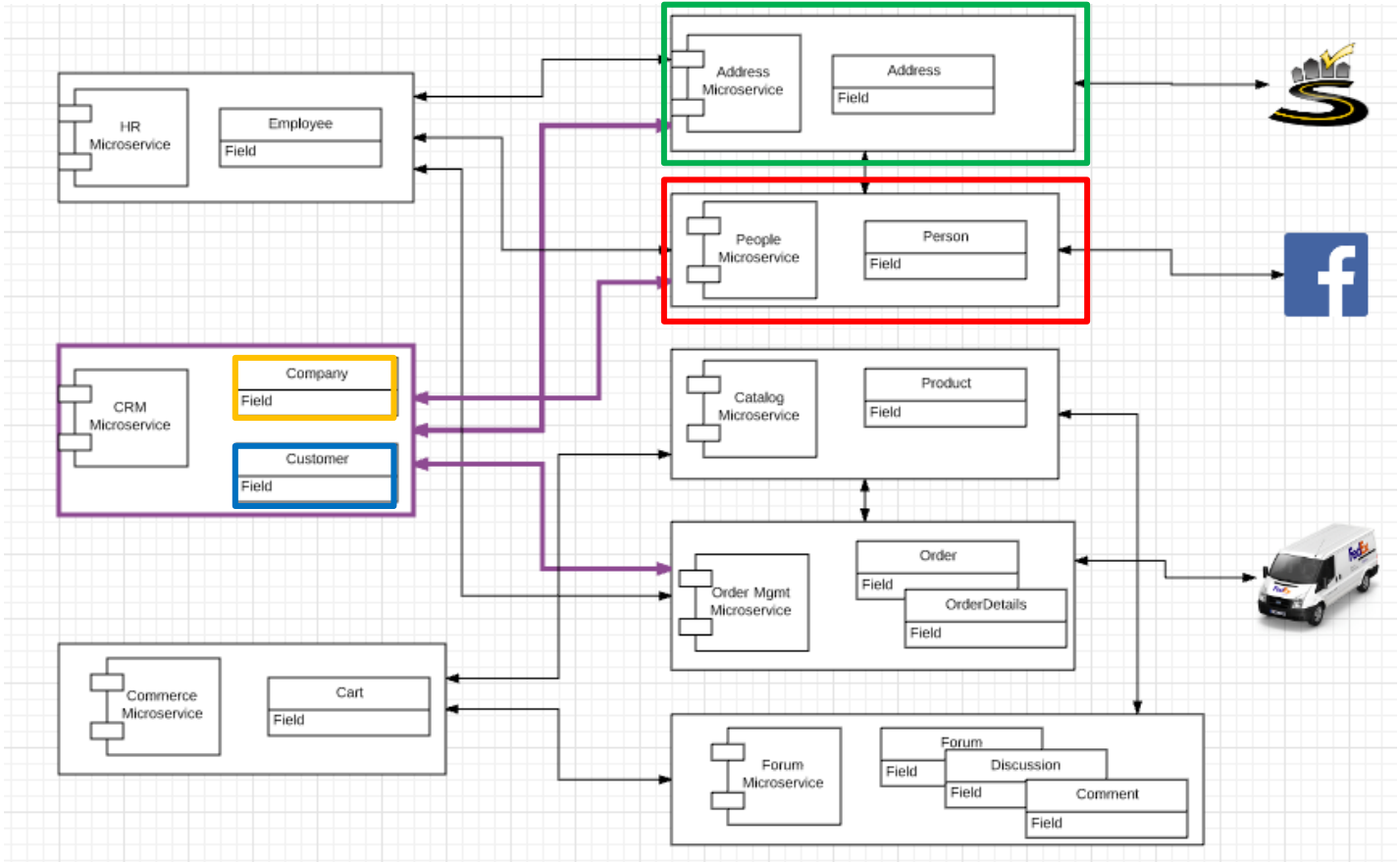
```
{
   "name" : "Bob's Widgets",
   "dunsNo" : "189948347",
   "address" : {
      "street1" : "1234 Soth Street",
      "city" : "Boston",
      "region" : "MA",
      "country" : "US",
      "postalCode" : "12345"
   },
   "contact" : {
      "lastName" : "Robertson",
      "firstName" : "Robert",
      "title" : "VP, Purchasing"
   }
}
```



Our Modified Data Model



Orchestration

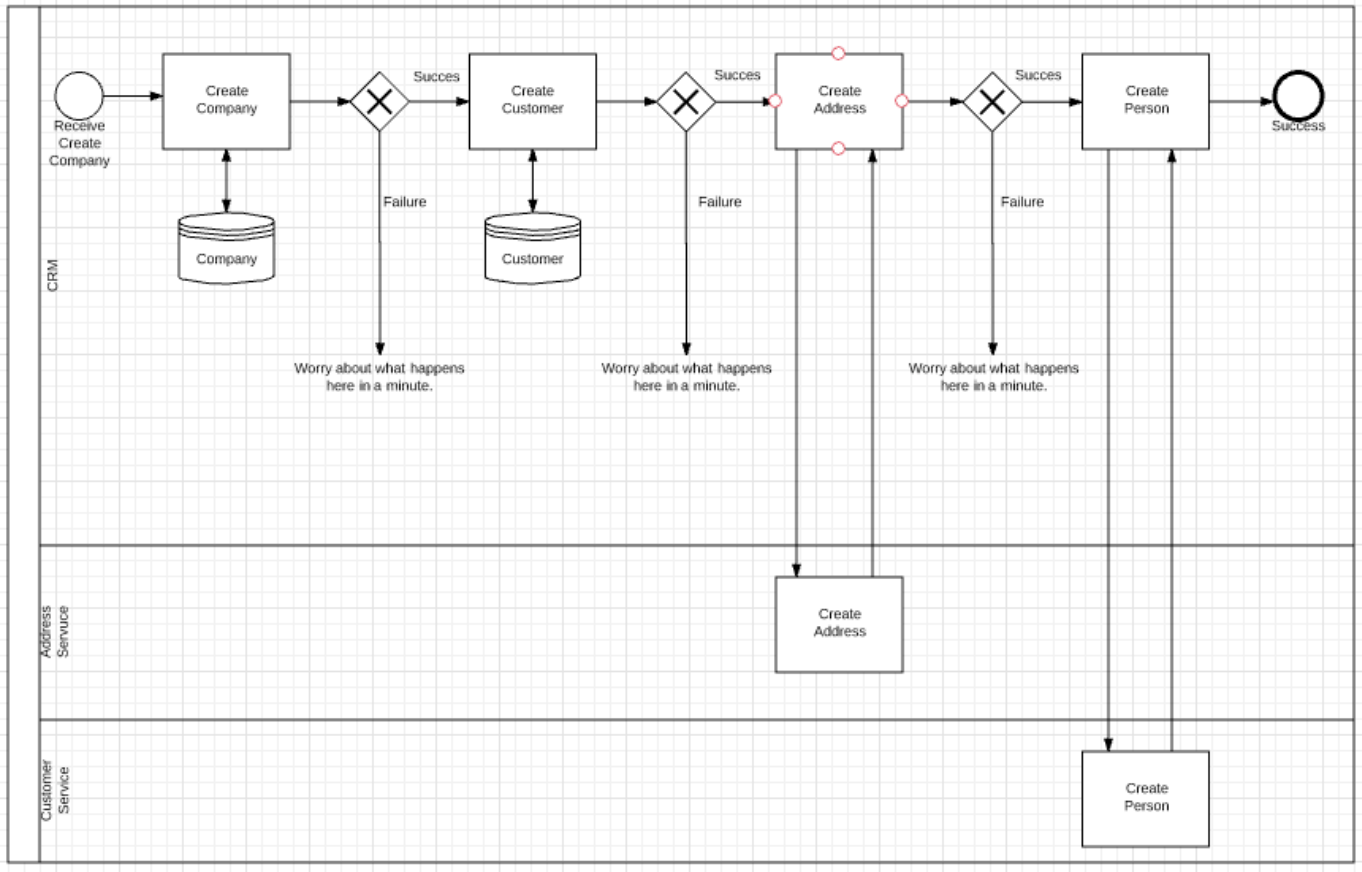


Comments

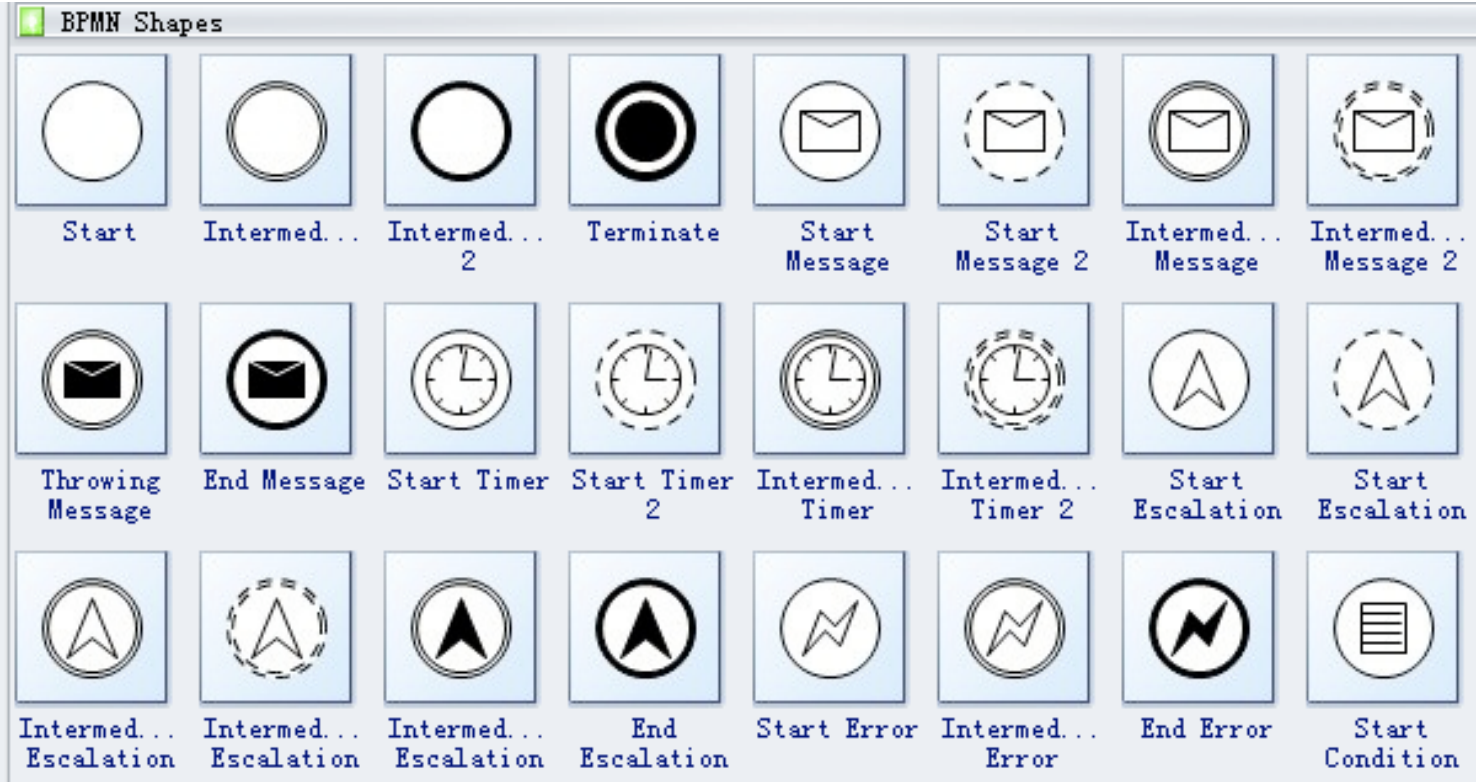
- The datamodel is contrived and "over normalized."
  - Each data element occurs in EXACTLY one table/microservice.
  - Normally, a data element has instances/copies in several locations.
  - We are currently focusing on core concepts.
  - Will handle data integration/synchronization in later lectures.

- Notice that in this pattern, POST creates the URL of the form xxx/id. This example shows 3 patterns
  - Use an intrinsic property of the data, e.g. SSNO, DUNS No.
  - Generate, e.g. UUID, Auto-Increment.
  - Simple algorithm applied to the non-unique data to make something unique.
    - Your UNIs are an example.
    - Users are
      - Capable of entering data in the "Please enter your UNI" box, but
      - Would be unhappy if every application asked for some complex, random UUID.

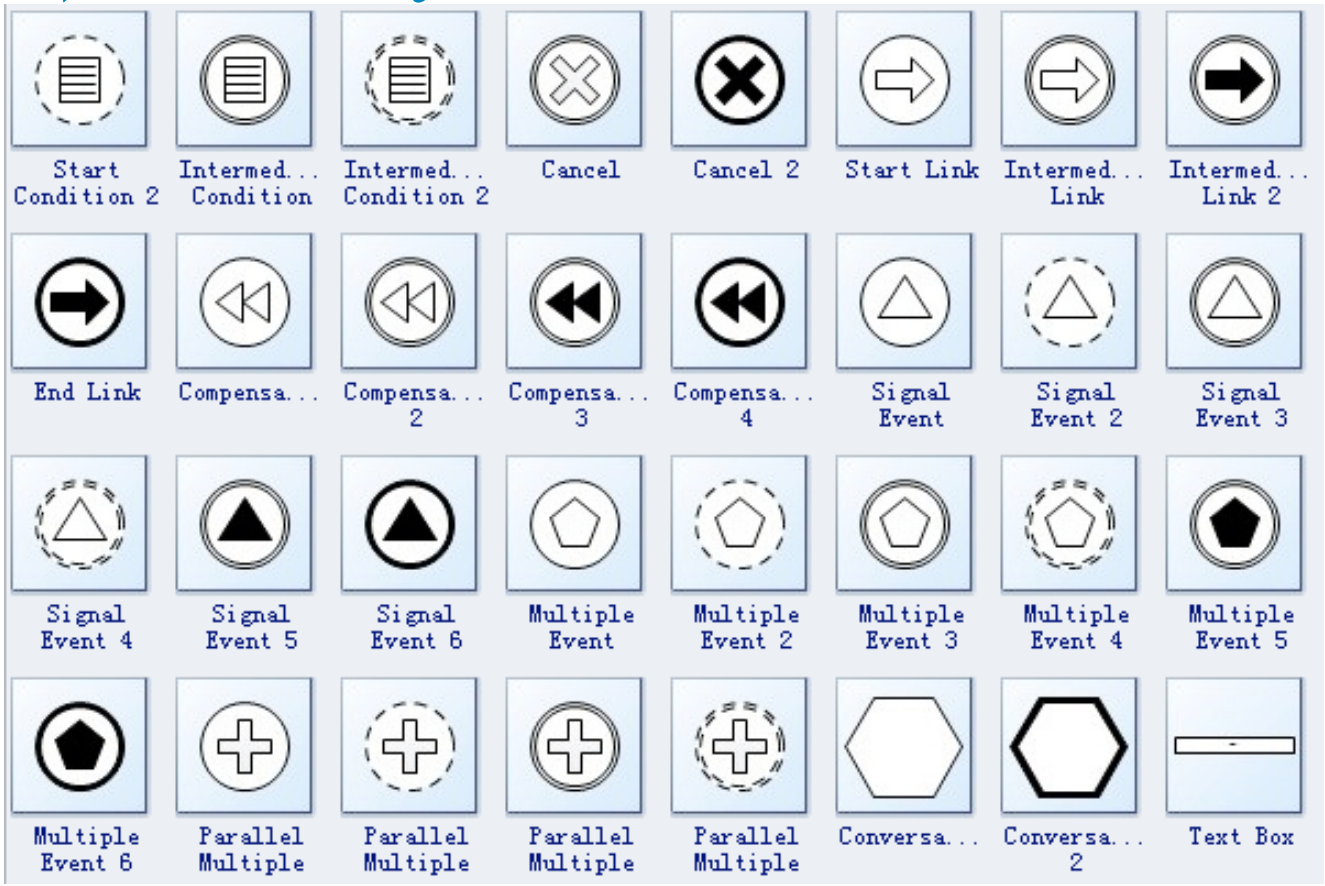
Orchestration (BPMN Diagram)



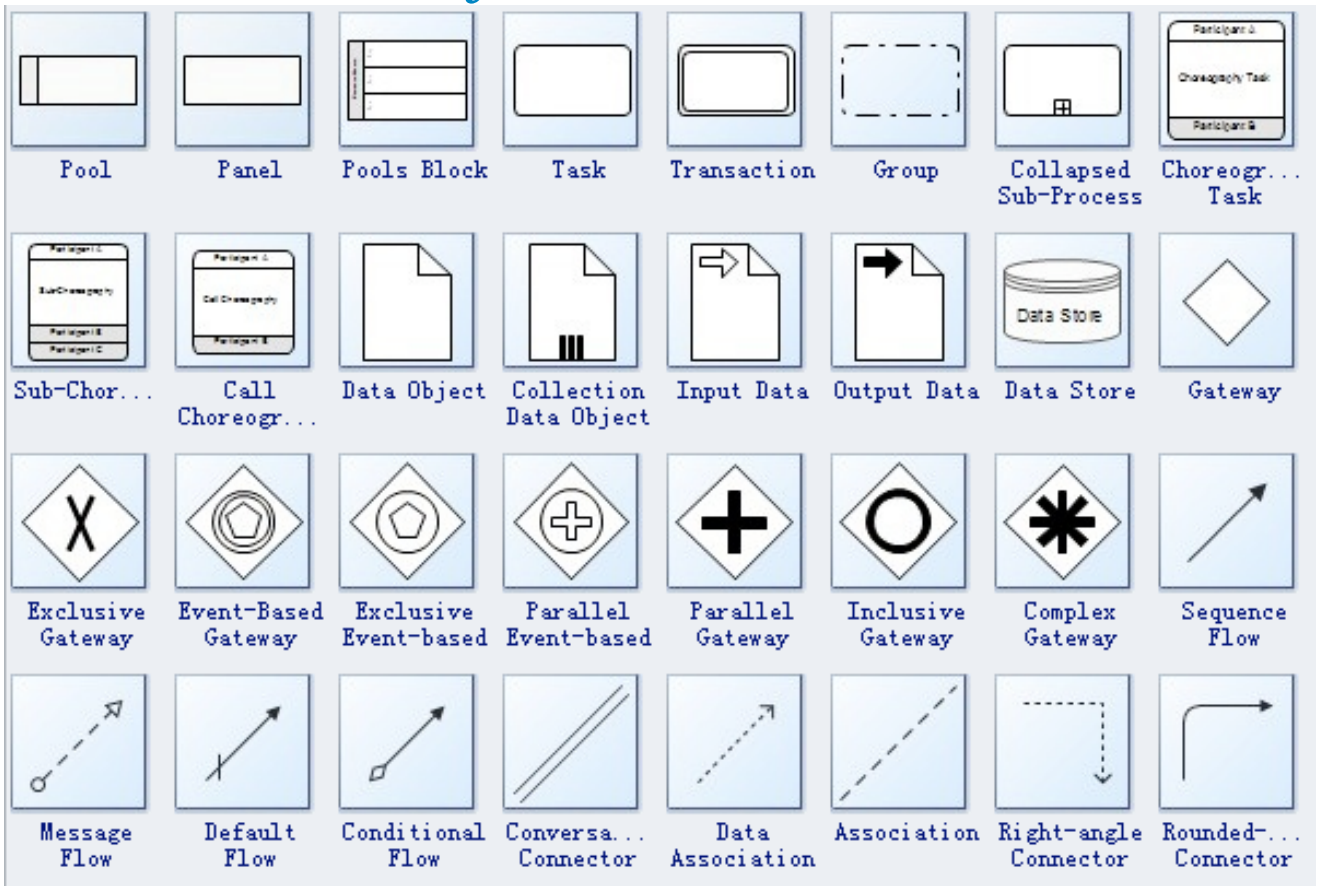
(Some) BPMN Symbols/Notation



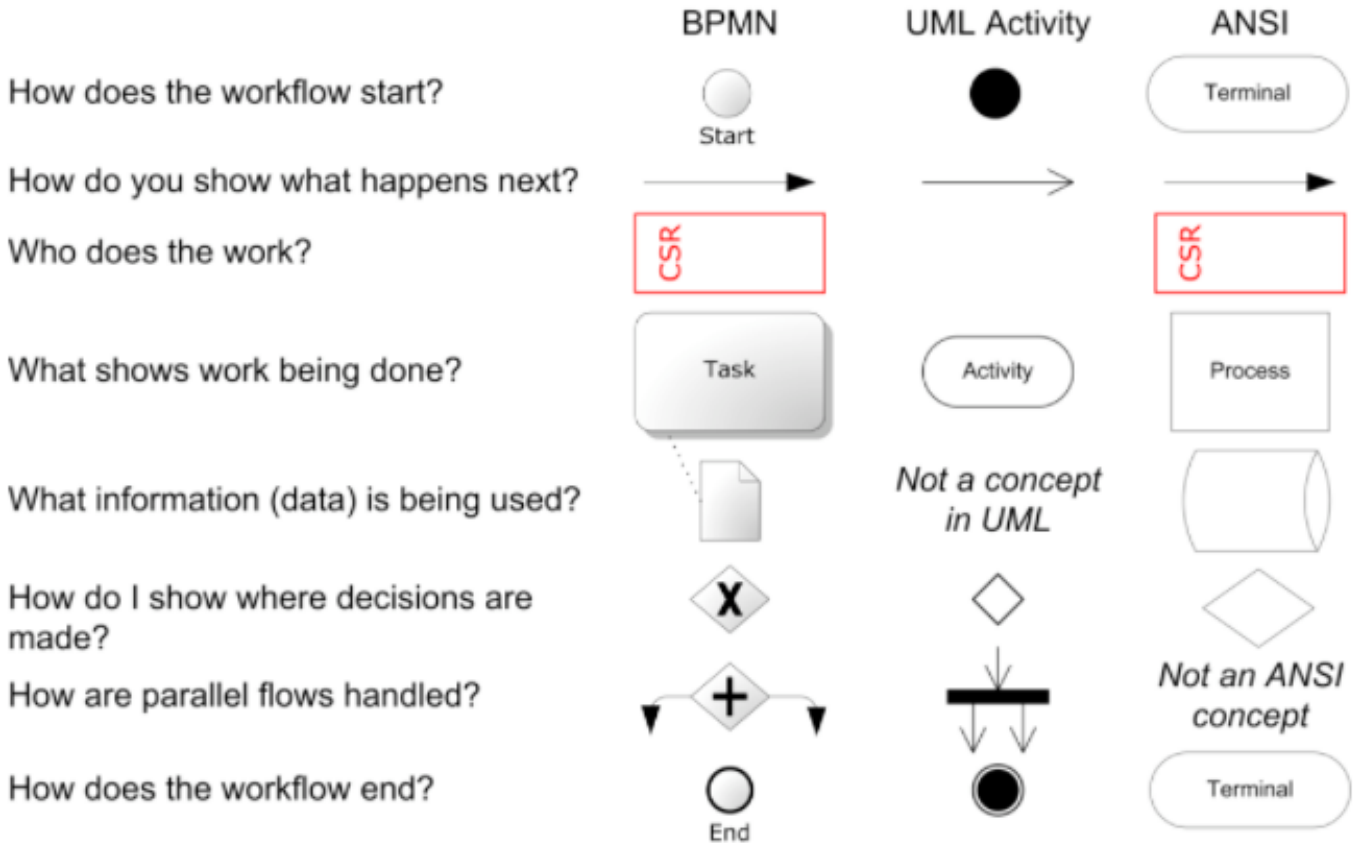
(Some) BPMN Symbols



Some BPMN Symbols



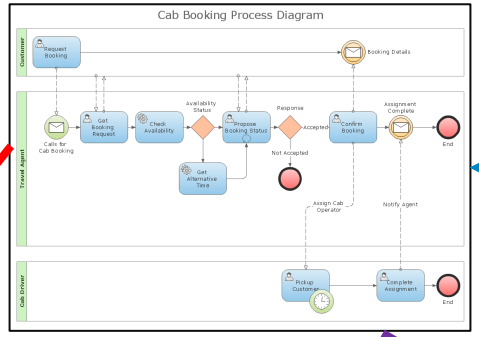
There are Many Notations



Flow Diagrams: 3 Perspectives

1. Use like PPT to *explain/document* how an application/environment functions.

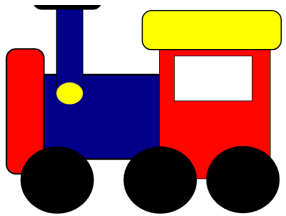
2. Use like PPT to *design/explain* some complex code I am going to develop.




```
var app = angular.module('app', []);
app.controller('SampleCtrl', function($scope, $q){
    $scope.fail = false;
    $scope.test = function(){

        var deferred = $q.defer();

        var promise = deferred.promise;
        promise.then(function(result){
            alert('Success: ' + result);
        }, function(reason){
            alert('Error: ' + reason);
        });

        if($scope.fail)
            deferred.reject('sorry');
        else
            deferred.resolve('Cool');
    };
});
```

3. Emit an executable representation of the flow.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<bpmnQuestionDataStorage>
    <dataStorage xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="bpmnDataStorage">
        <errorelement id=1>
            <element id=6 />
            <descr>The gateway semantically does not match its opening counterpart.</descr>
        </errorelement>
    </dataStorage>
    <question>
        <task>
            Please check the BPMN model shown below for syntactic or semantic errors.
        </task>
        <model path="./bpmn_test_1.png">
            <element id=1 type="ev-st" x=21 y=114 w=48 h=48 />
            <element id=2 type="task" x=106 y=90 w=125 h=100 />
            <element id=3 type="gw-op" x=272 y=114 w=48 h=48 />
            <element id=4 type="task" x=340 y=17 w=125 h=100 />
            <element id=5 type="task" x=340 y=161 w=125 h=100 />
            <element id=6 type="gw-cl" x=504 y=114 w=48 h=48 />
            <element id=7 type="task" x=572 y=90 w=125 h=100 />
            <element id=8 type="ev-en" x=732 y=114 w=48 h=48 />
        </model>
        <supporttext>
            If you found an error, please mark it by clicking on it.
        </supporttext>
    </question>
    <difficulty>0.74</difficulty>
</bpmnQuestionDataStorage>
```



Flow Diagrams: 3 Perspectives

1. Use like PPT to *explain/document* n application/environment functions.

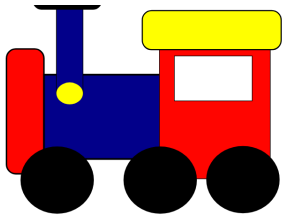
2. Use *desi* some c I am goi

- (1) and (2) are probably familiar to you.
- (3) is probably an unfamiliar concept. So, let's take a quick look.

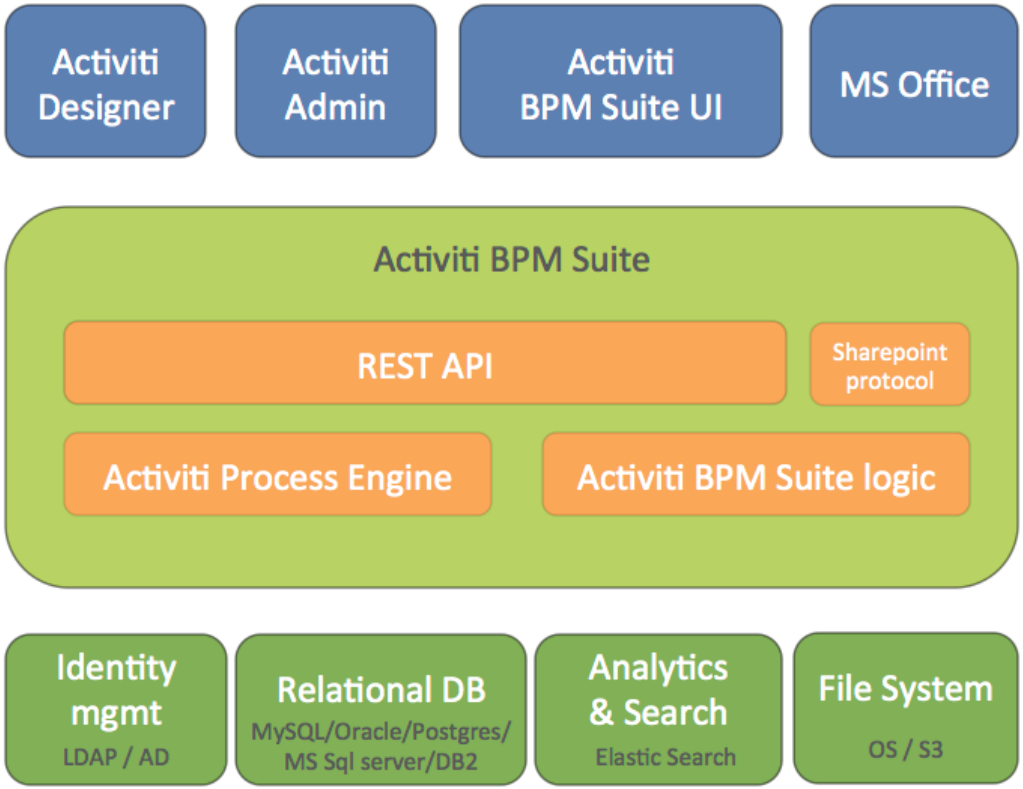
3. Emit an executable representation of the flow.

```
var app = angular.module('app', []);
app.controller('SampleCtrl', function($scope, $q){
    $scope.fail = false;
    $scope.test = function(){

        var deferred = $q.defer();

        var promise = deferred.promise;
        promise.then(function(result){
            alert('Success: ' + result);
        }, function(reason){
            alert('Error: ' + reason);
        });

        if($scope.fail)
            deferred.reject('sorry');
        else
            deferred.resolve('Cool');
    };
});
```

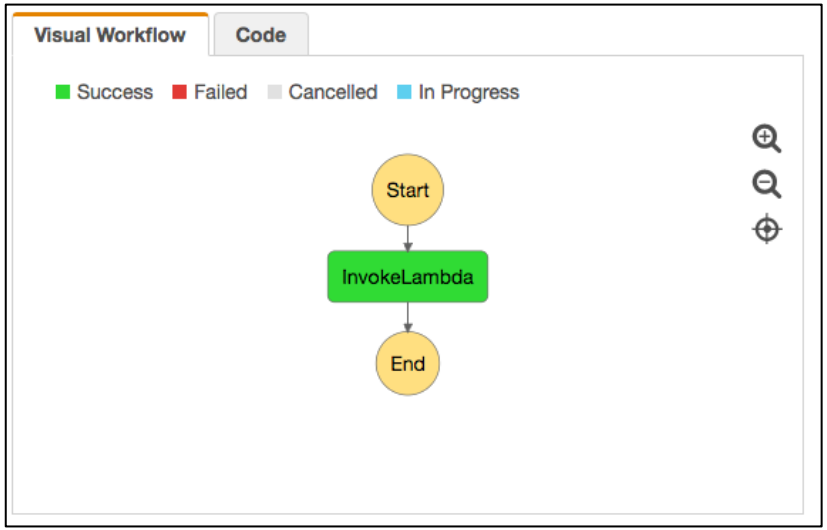
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<bpmnQuestionDataStorage>
    <dataStorage xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="bpmnDataStorage">
        <errorelement id=1>
            <element id=6 />
            <descr>The gateway semantically does not match its opening counterpart.</descr>
        </errorelement>
    </dataStorage>
    <question>
        <task>
            Please check the BPMN model shown below for syntactic or semantic errors.
        </task>
        <model path="./bpmn_test_1.png">
            <element id=1 type="ev-st" x=21 y=114 w=48 h=48 />
            <element id=2 type="task" x=106 y=90 w=125 h=100 />
            <element id=3 type="gw-op" x=272 y=114 w=48 h=48 />
            <element id=4 type="task" x=340 y=17 w=125 h=100 />
            <element id=5 type="task" x=340 y=161 w=125 h=100 />
            <element id=6 type="gw-cl" x=504 y=114 w=48 h=48 />
            <element id=7 type="task" x=572 y=90 w=125 h=100 />
            <element id=8 type="ev-en" x=732 y=114 w=48 h=48 />
        </model>
        <supporttext>
            If you found an error, please mark it by clicking on it.
        </supporttext>
    </question>
    <difficulty>-0.74</difficulty>
</bpmnQuestionDataStorage>
```



Example: Alfresco Activiti BPM Suite



AWS Step Functions and Workflow



Visual Workflow | Code

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

AWS Step Functions and Workflow

Visual Workflow | Code

```
{
  "Comment": "A Hello World example of the Amazon States Language using an AWS Lambda Function",
  "StartAt": "InvokeLambda",
  "States": {
    "InvokeLambda": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:832720255830:function:helloworld",
      "End": true
    }
  }
}
```

Execution Details

Info | Input | Output

Execution Status
Succeeded

State Machine Arn
arn:aws:states:us-east-1:832720255830:stateMachine:columbiasteptest

Execution ID
arn:aws:states:us-east-1:832720255830:execution:columbiasteptest:e7ac0395-3c6d-11e1-96b1-245699df7d0d

Started
Sep 24, 2017 09:06:44 AM

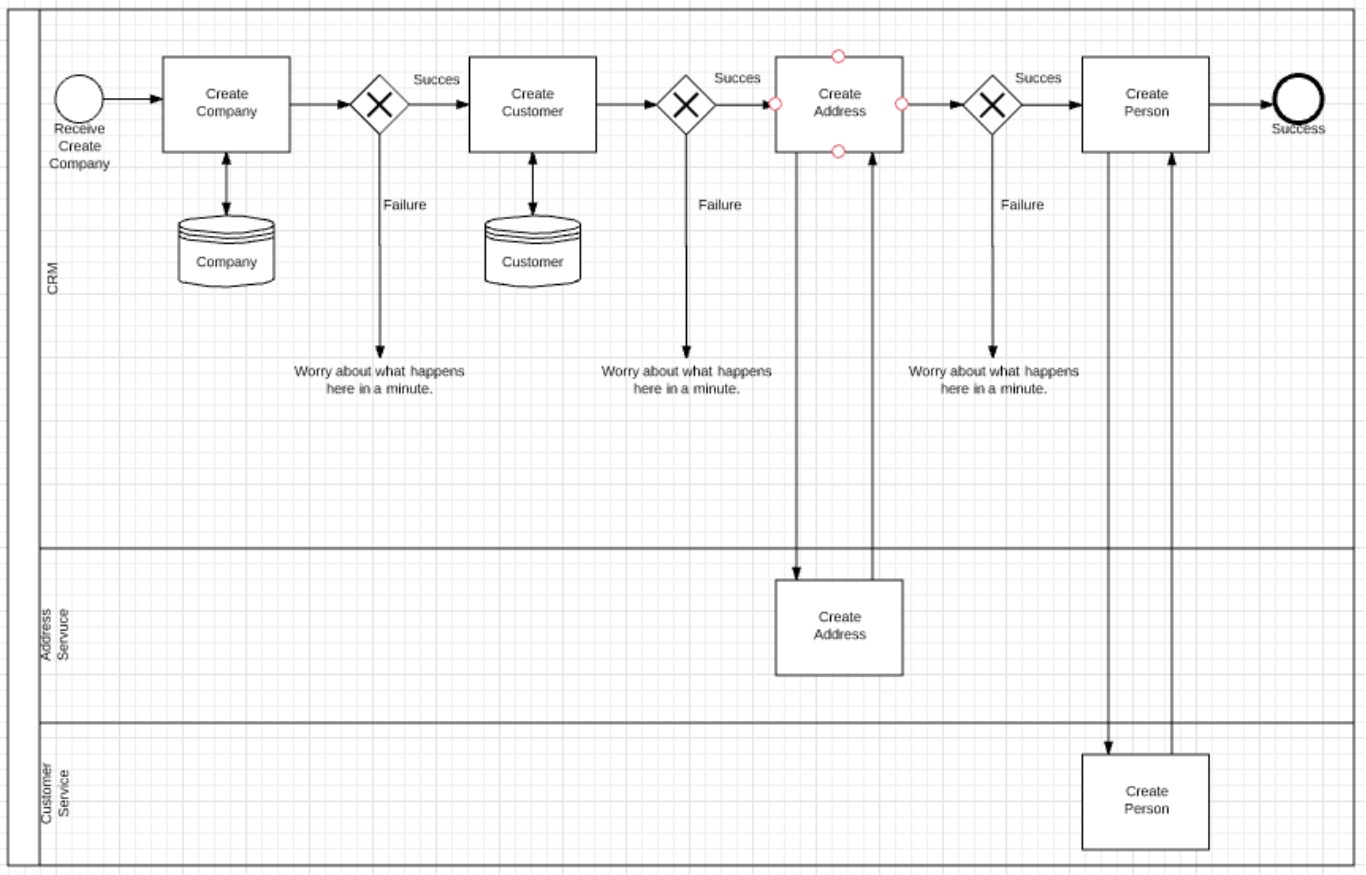
Closed
Sep 24, 2017 09:06:44 AM

Step Details

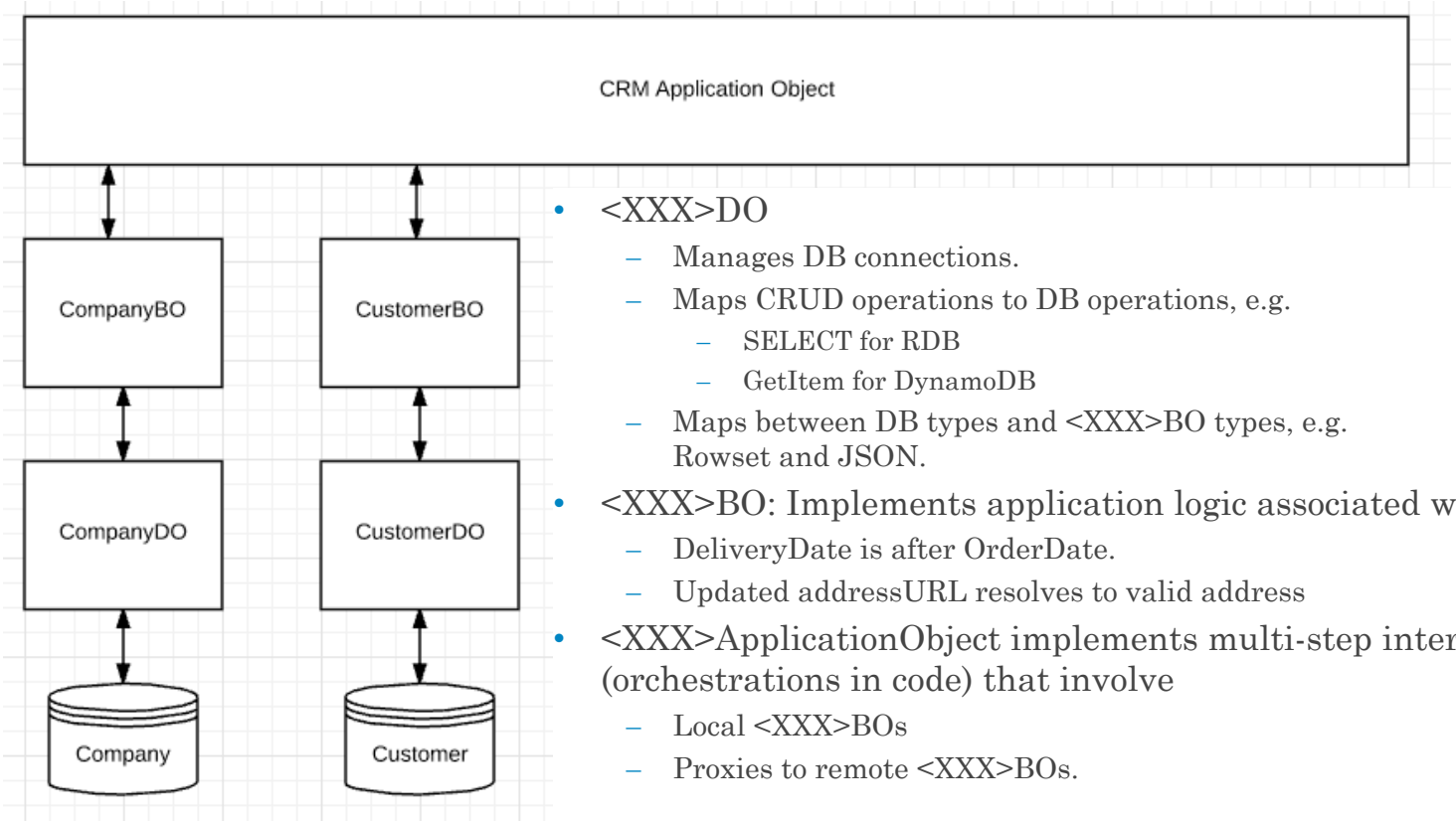
ID	Type	Timestamp
1	ExecutionStarted	Sep 24, 2017 09:06:44 AM
2	TaskStateEntered	Sep 24, 2017 09:06:44 AM
3	LambdaFunctionScheduled	Sep 24, 2017 09:06:44 AM

Implement with Code (Will come back to Engines)

Design and Implement with Code

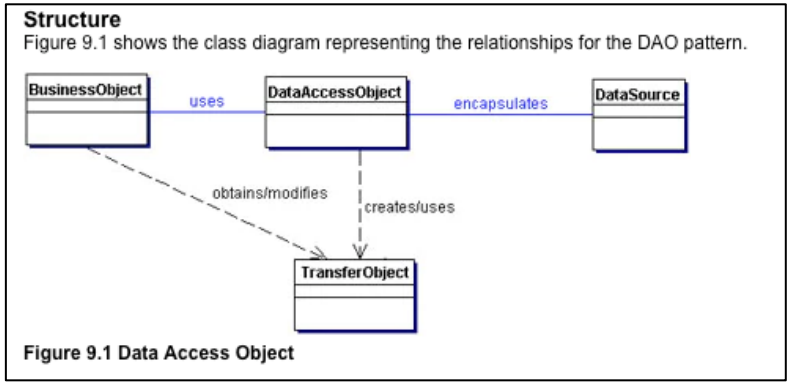


Implementation Pattern – Part I

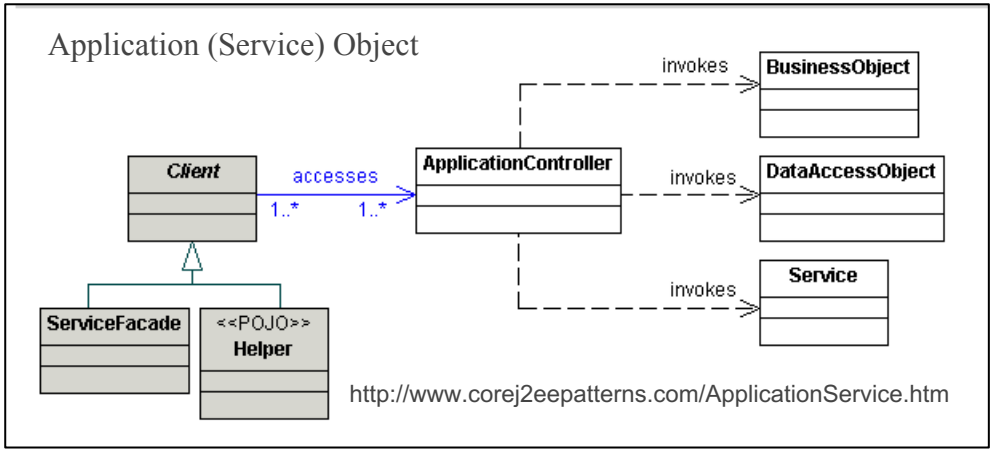


- <XXX>DO
  - Manages DB connections.
  - Maps CRUD operations to DB operations, e.g.
    - SELECT for RDB
    - GetItem for DynamoDB
  - Maps between DB types and <XXX>BO types, e.g. Rowset and JSON.
- <XXX>BO: Implements application logic associated with an entity, e.g.
  - DeliveryDate is after OrderDate.
  - Updated addressURL resolves to valid address
- <XXX>ApplicationObject implements multi-step interactions (orchestrations in code) that involve
  - Local <XXX>BOs
  - Proxies to remote <XXX>BOs.

Implementation Pattern – Part I



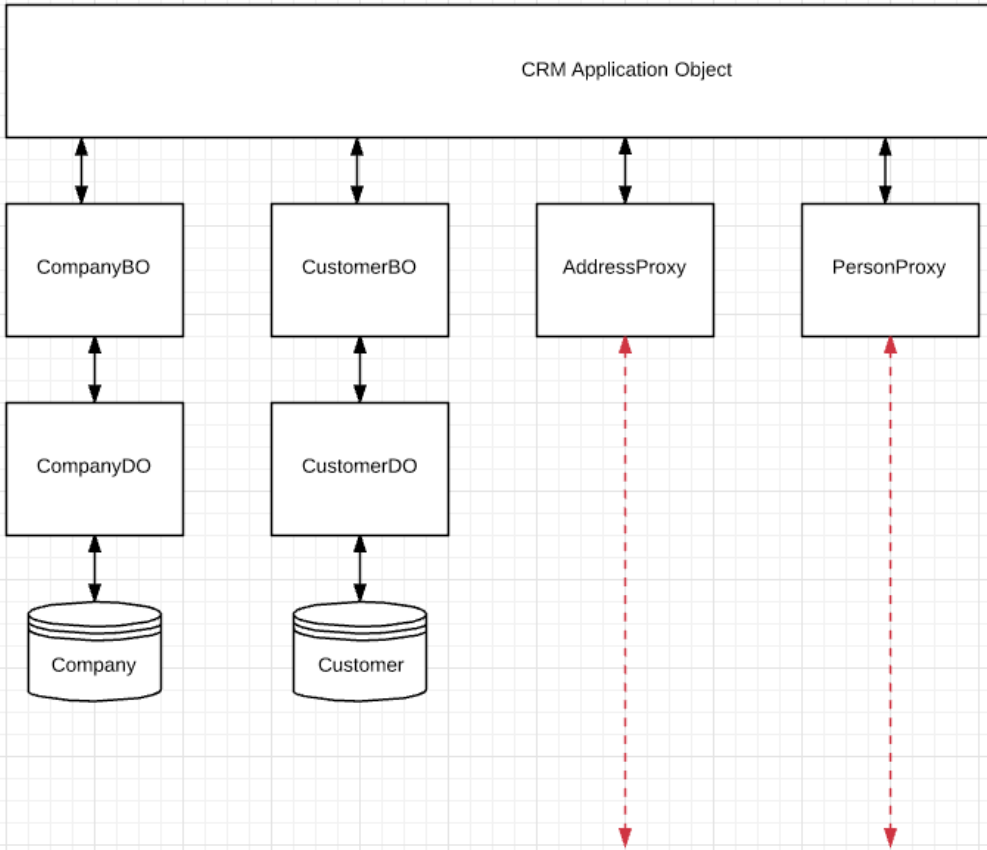
http://www.oracle.com/technetwork/java/dataaccessobject-138824.html



http://www.corej2eepatterns.com/ApplicationService.htm

- Despite the adoption of new technology and models, e.g.
  - XaaS, Cloud
  - Microservices
- The same basic application design best practices apply.
- This is one pattern. Do not get hung up on which patterns among many choices, but make sure that you leverage the vast experience codified in patterns.

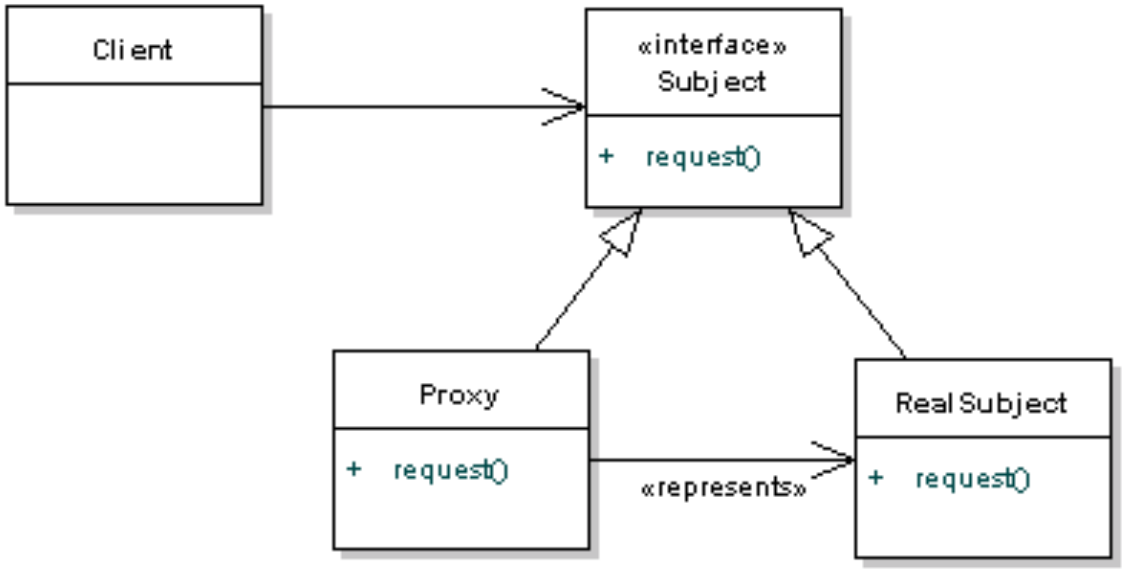
Implementation Pattern – Part II



- Same interface as the <XXX>BO
- Calls the actual implementation.

Remote (HTTP) Calls

Proxy Pattern (Example Reference)



https://dzone.com/articles/design-patterns-proxy

Company Data Object

```
// jshint node: true

var db = require('./lib/db');
var logging = require('./lib/logging');


var createIdFromName = function(name) {...};

exports.create = function(companyTO) {...};

exports.get_by_id = function(id) {...};

exports.get_by_template = function(companyTOTemplate) {...};

exports.update_by_id = function(id, newCompanyTO) {...};

exports.delete_by_id = function(id) {...};
```

Company Data Object

```
exports.create = function(companyTO) {

    return new Promise(function (resolve, reject) {
        var id;


        id = createIdFromName(companyTO.name);

        var sql = "insert into e6998.Company values(" +
            "'" + id + "', " +
            "'" + companyTO.name + "'," +
            "'" + companyTO.address + "', " +
            "'" + companyTO.contact + "');"

        logging.debug_message("Create: SQL = " + sql);

        db.execute_query(sql, null).then(
            function (result) {
                logging.debug_message("Create result =", result);
                resolve(result);
            },
            function (error) {
                logging.debug_message("Create error = ", error);
                resolve(error);
            });
    });
};
```

Not great code,
but you get the idea.

The SQL-ness of it
is not visible to caller.

Test CompanyDO Twice

```
var CompanyDO = require('./CompanyDO');
var logging = require('./lib/logging');

CompanyDO.create({name: "Don Company", address: "/api/addresses/102", contact: "/api/contacts/345"}).then(
    function(result) {
        logging.debug_message("*********************************************");
        logging.debug_message ("DO layer returned ", result);
        logging.debug_message("*********************************************")
    },
    function(error) {
        logging.debug_error("*********************************************")
        logging.error_error("DO Layer returned ERROR = ", error);
    }
);
```

```
*****************************************
DO layer returned  {
  "fieldCount": 0,
  "affectedRows": 1,
  "insertId": 0,
  "serverStatus": 2,
  "warningCount": 0,
  "message": "",
  "protocol41": true,
  "changedRows": 0
}
*****************************************
```

1st Time

```
*********************************************
DO layer returned  {
  "code": "ER_DUP_ENTRY",
  "errno": 1062,
  "sqlMessage": "Duplicate entry 'DONC' for key 'PRIMARY'",
  "sqlState": "23000",
  "index": 0,
  "sql": "insert into e6998.Company values('DONC', 'Don Company','/api/addresses/102', '/api/contacts/345');"
}
*********************************************
```

2nd Time

Test CompanyDO Twice

```
var CompanyDO = require(
var logging = require(

CompanyDO.create({nam
    function(result)
        logging.debug
        logging.debug
        logging.debug
    },
    function(error) {
        logging.debug
        logging.error
    }
);
```

- The SQL-ness shows through.
- There should be a standard architecture for DO-layer
  - Return codes and messages.
  - Data formats
- Regardless of the specific DB engine.

```
****************************************
DO layer returned  {
  "fieldCount": 0,
  "affectedRows": 1,
  "insertId": 0,
  "serverStatus": 2,
  "warningCount": 0,
  "message": "",
  "protocol41": true,
  "changedRows": 0
}
****************************************
```

1st Time

```
************************************************
DO layer returned  {
  "code": "ER_DUP_ENTRY",
  "errno": 1062,
  "sqlMessage": "Duplicate entry 'DONC' for key 'PRIMARY'",
  "sqlState": "23000",
  "index": 0,
  "sql": "insert into e6998.Company values('DONC', 'Don Company','/api/addresses/102', '/api/contacts/345');"
}
************************************************
```

2nd Time

CompanyBO

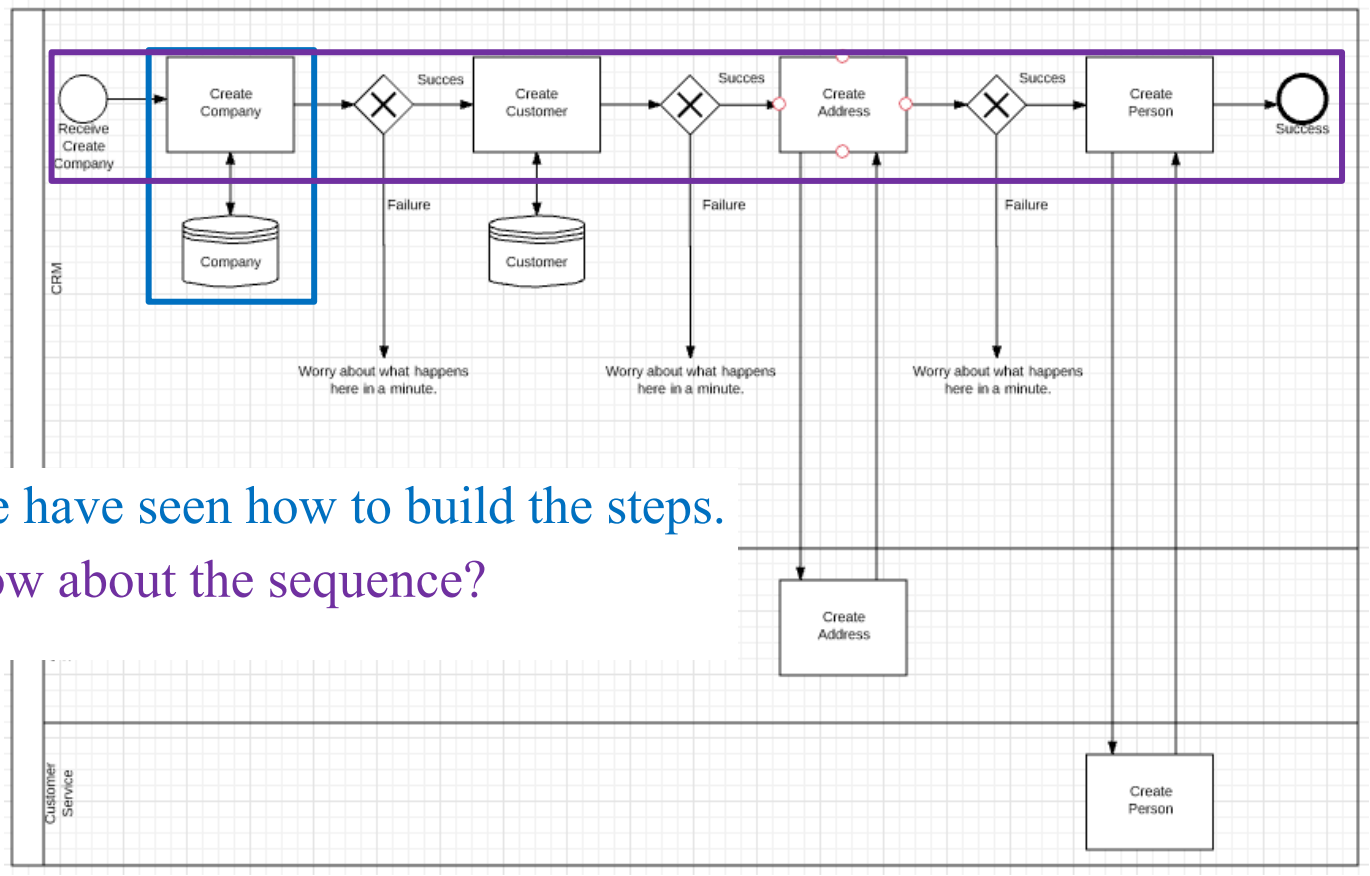
```
var CompanyDO = require('./CompanyDO');
var logging = require('./lib/logging');


exports.create = function(companyTO) {...};

exports.get_by_id = function(id) {...};

exports.get_by_template = function(companyTOTemplate) {...};

exports.update_by_id = function(id, newCompanyTO) {...};

exports.delete_by_id = function(id) {...};
```

CompanyBO

- Pretty straightforward
- A real application would have more complicated code
  - Precondition validation
  - Post-condition
    - Validation
    - Side effects
  - Error recovery/retry.
- Pattern is very similar for
  - All methods on a BO
  - All BOs.

```
exports.create = function(companyTO) {

    return new Promise(function (resolve, reject) {

        // DO some validation, pre-processing, before conditions
        var preconditionSuccess = false;

        if (companyTO.name) {
            preconditionSuccess = true;
        }

        if (preconditionSuccess) {
            CompanyDO.create(companyTO).then(
                function (result) {
                    logging.debug_message("*****************************************");
                    logging.debug_message("DO layer returned ", result);
                    logging.debug_message("*****************************************")

                    // Do some post processing conditions and normalize the result.
                    resolve(result);
                },
                function (error) {
                    logging.debug_error("*****************************************")
                    logging.error_error("DO Layer returned ERROR = ", error);
                    logging.debug_error("*****************************************")

                    // Do some post processing conditions and normalize the error.
                    resolve(result);
                }
            );
        }
        else {
            logging.debug_error("*****************************************")
            logging.error_error("Some precondition failed;");
            logging.debug_error("*****************************************")

            // Do some post processing conditions and normalize the error.
            reject("Precondition failed and useful error data");
        }
    });
};
```

Orchestration (BPMN Diagram)



- We have seen how to build the steps.
- How about the sequence?

CRMService

```
var CompanyBO = require('./CompanyBO');
var ContactBO = require('./ContactBO');
var PersonBO = require('./PersonProxyBO');
var AddressBO = require('./AddressProxyBO');

var logging = require('./lib/logging');


exports.createCompany = function(createCompanyReq) {...};

// Other functions go here, for example creating
// other more complex, multi-object things.
```

Wait!

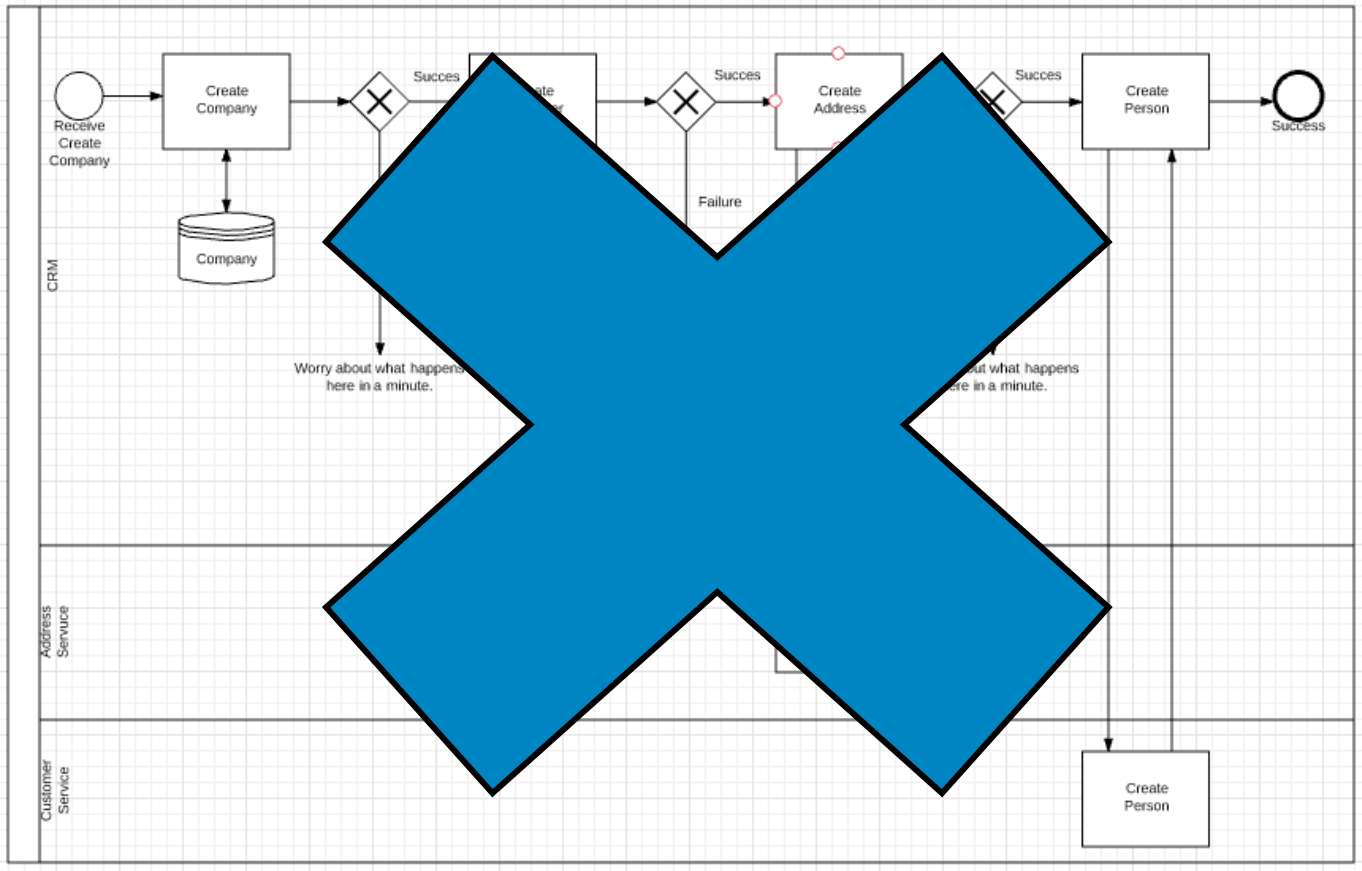
I need the

- Address and the Contact URLs before I can create the Company.
- And the Person before I can create the Contact

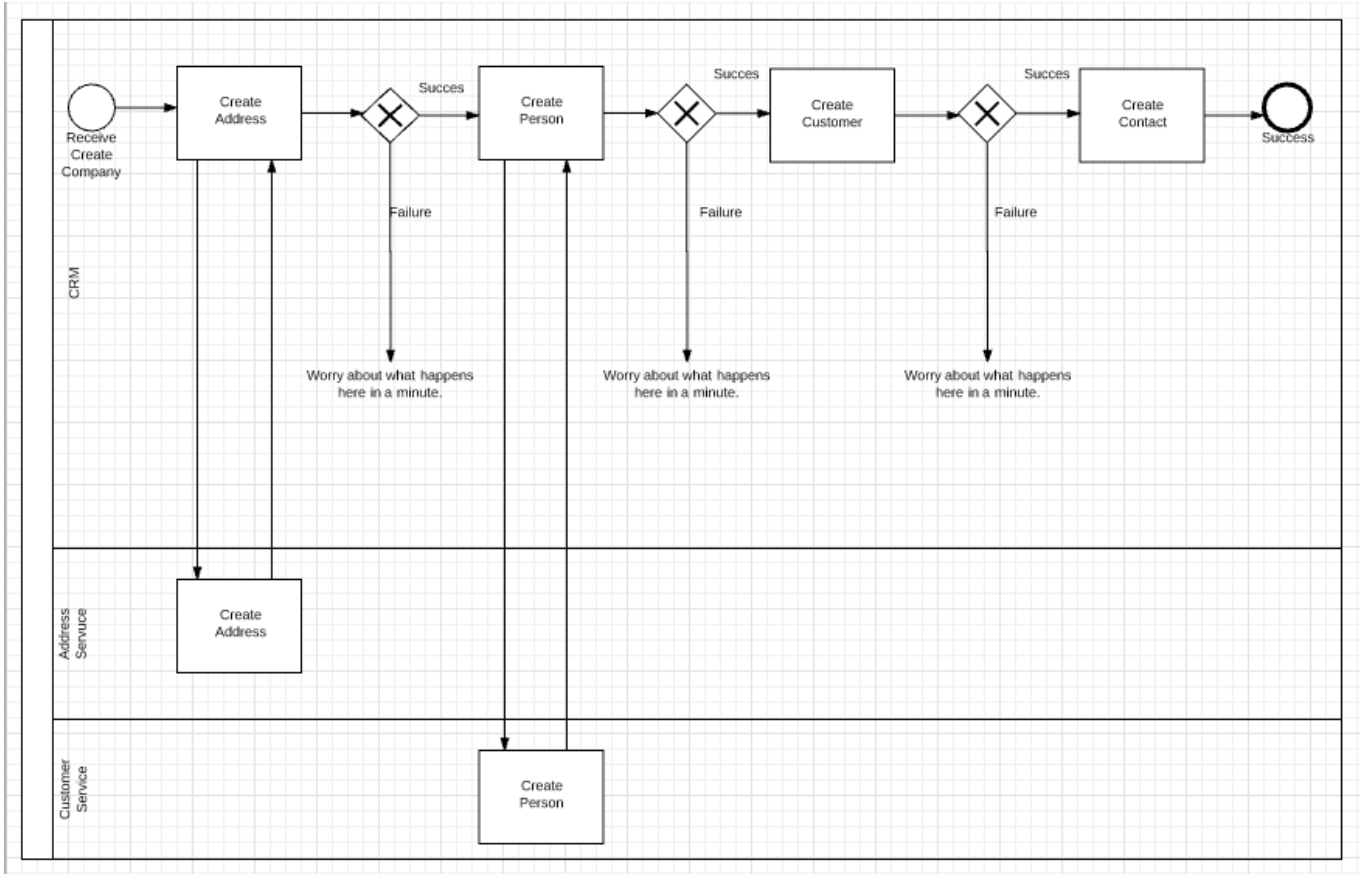
- Would prefer not to have to hardcode knowledge of using BO or Proxy.
- There is a Service Locator pattern for this situation.
- The code would
  - Require the Service Locator
  - Which would know whether to call was local or remote.

```
{
    "name" : "Bob's Widgets",
    "dunsNo" : "189948347",
    "address" : {
        "street1" : "14 Beacon Street",
        "city" : "Boston",
        "region" : "MA",
        "country" : "US",
        "postalCode" : "02108"
    },
    "contact" : {
        "lastName" : "Robertson",
        "firstName" : "Robert",
        "title" : "VP, Purchasing"
    }
}
```

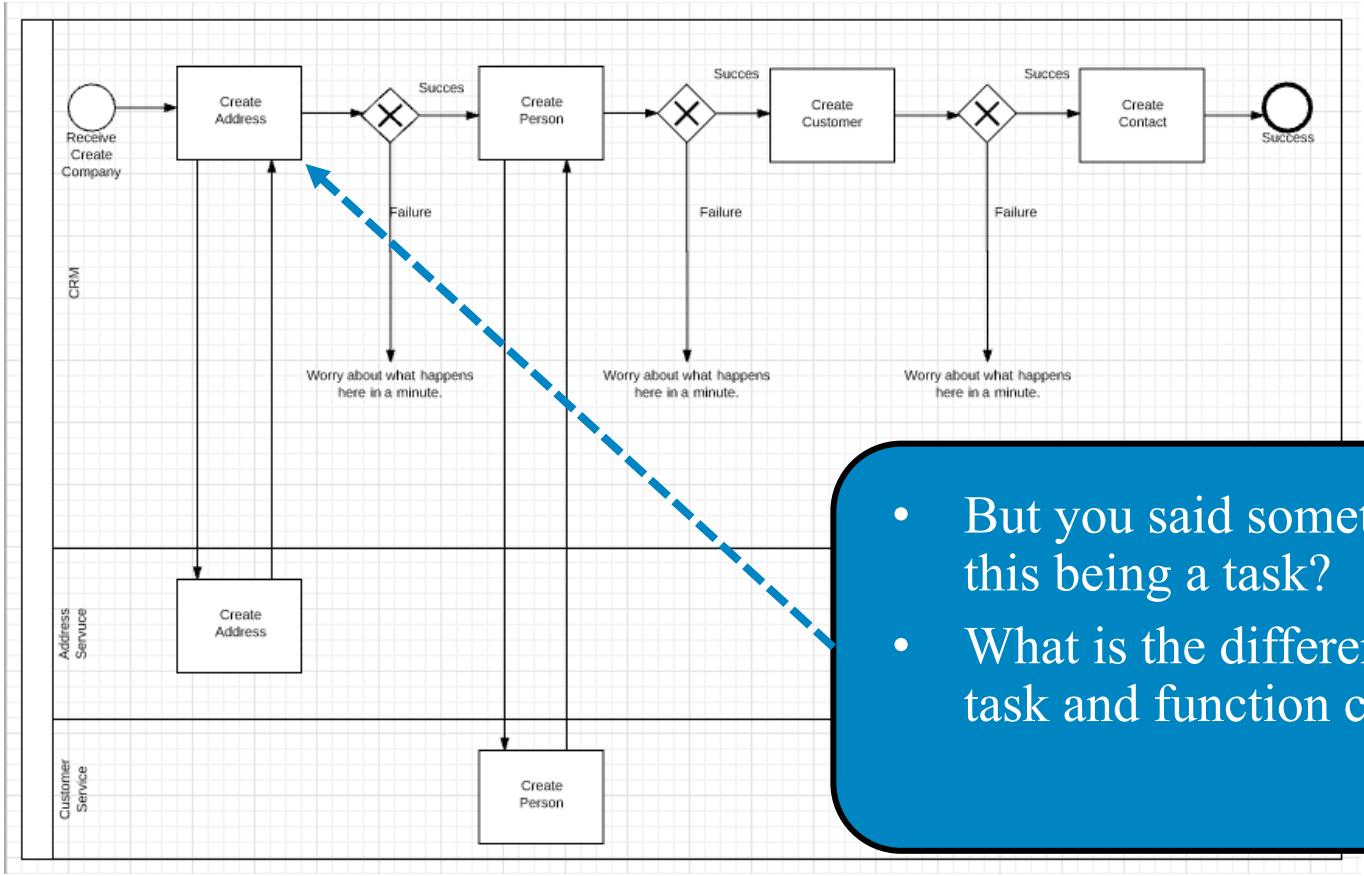
Epic Fail!



Maybe this is Better?



Maybe this is Better?



- But you said something about this being a task?
- What is the difference between a task and function call?

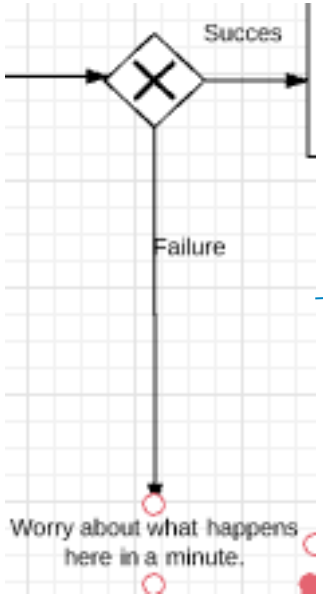
Task Versus Function Call

The task

- Validates that the task input is correct.
- Extracts the data needed for the function call.
- Calls the function.
- Validates the result.
- Adds the new/updated data into the overall request.

```
var createAddressTask = function(createCompanyReq) {

    return new Promise(function (resolve, reject) {

        // Do some validation
        if (!valid) {
            Promise.reject("Some meaningful error condition");
            return;
        }

        AddressBO.create(createCompanyReq.address).then(
            function (result) {
                // A well-defined POST/CREATE returns the URL to the thing created.
                createCompanyReq.address.links.self =
                    {
                        href: result.url
                    };
                createCompanyReq.address = createCompanyReq.address.links.self;

                // I have updated the create request flowing through the pipeline
                // with additional information needed downstream.
                resolve(createCompanyReq);
            },
            function (error) {
                // Do some recovery.
                //
                reject(error);
            });
    });
};
```

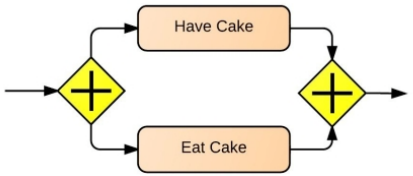
CRM Service createCompany Psuedocode




```
exports.createCompany = function(createCompanyReq) {

    return new Promise(function (resolve, reject) {

        // DO some validation, pre-processing, before conditions
        var preconditionSuccess = validateCreateCompanyReq();        // This may be

        if (!preconditionSuccess) {
            Promise.reject("Some meaningful message");
        }

        createAddressTask(createCompanyReq)
            .then(createPersonTask, someErrorHandler)
            .then(createCompanytask, anotherErrorHandler)
            .then(createContactTask, thirdErrorHandler)
            .then(
                function(result) {
                    // We can return
                    resolve(result);
                },
                function(error) {
                    // Better trigger some undo behavior here.
                    undoBehavior(createCompanyReq);
                })
            .catch(finalErrorHandler);
    });
};
```

Task (Promise) Composition Patterns

- Chain
  - Run one at a time until end or failure
  - A.then(b).then(c).then(d).then(e).then()
  - Different libraries have slightly formats for then(), catch(), finally(), etc.
  - You can build the chain
    - Explcitly concatenating in code or
    - By building an array.
- Parallel
  - Promise.all().then(f1, f2)
  - f1 receives an array of the promise resolution on success.
  - f2 called when the first promise fails.
- Race/Any
- etc.



The actual process/service is combinations of combinations.

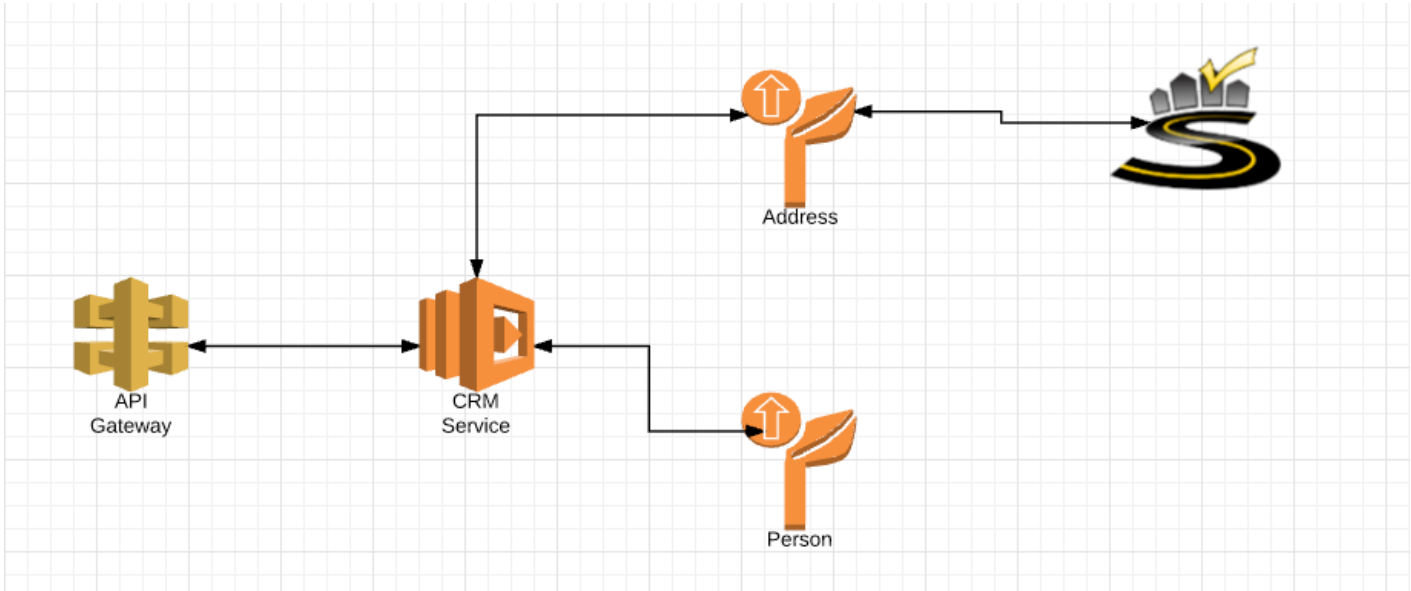
Beyond Simple Promises (http://bluebirdjs.com/docs/api-reference.html)

API Reference

- Core
  - new Promise
  - .then
  - .spread
  - .catch
  - .error
  - .finally
  - .bind
  - Promise.join
  - Promise.try
  - Promise.method
  - Promise.resolve
  - Promise.reject
- Synchronous inspection
  - PromiseInspection
  - .isFulfilled
  - .isRejected
  - .isPending
  - .isCancelled
  - .value
  - .reason
  - Promise.using
  - .disposer
- Promisification
  - Promise.promisify
  - Promise.promisifyAll
  - Promise.fromCallback
  - .asCallback
- Timers
  - .delay
  - .timeout
- Cancellation
  - .cancel
- Generators
  - Promise.coroutine
  - Promise.coroutine.addYieldHan
- Utility
  - .tap
  - .call
  - .get
  - .return
  - .throw
  - .reason
- Collections
  - Promise.all
  - Promise.props
  - Promise.any
  - Promise.some
  - Promise.map
  - Promise.reduce
  - Promise.filter
  - Promise.each
  - Promise.mapSeries
  - Promise.race
  - .all
  - .props
  - .any
  - .some
  - .map
  - .reduce
  - .filter
  - .each
  - .mapSeries
- Resource management
  - .throw
  - .catchReturn
  - .catchThrow
  - .reflect
  - Promise.getNewLibraryCopy
  - Promise.noConflict
  - Promise.setScheduler
- Built-in error types
  - OperationalError
  - TimeoutError
  - CancellationError
  - AggregateError
- Configuration
  - Global rejection events
  - Local rejection events
  - Promise.config
  - .suppressUnhandledRejections
  - .done
- Progression migration
- Deferred migration
- Environment variables

Lambda Functions

Next Project



Build on 1st to microservices

- Implement the CRM Service (and HR Service) using Lambda functions.
- Integrate with SmartyStreets

- Deliver web content via CloudFront and S3.

There are 5 principles of serverless architecture that describe how an ideal serverless system should be built. Use these principles to help guide your decisions when you create serverless architecture.

1. Use a compute service to execute code on demand (no servers)
2. Write single-purpose stateless functions
3. Design push-based, event-driven pipelines
4. Create thicker, more powerful front ends
5. Embrace third-party services

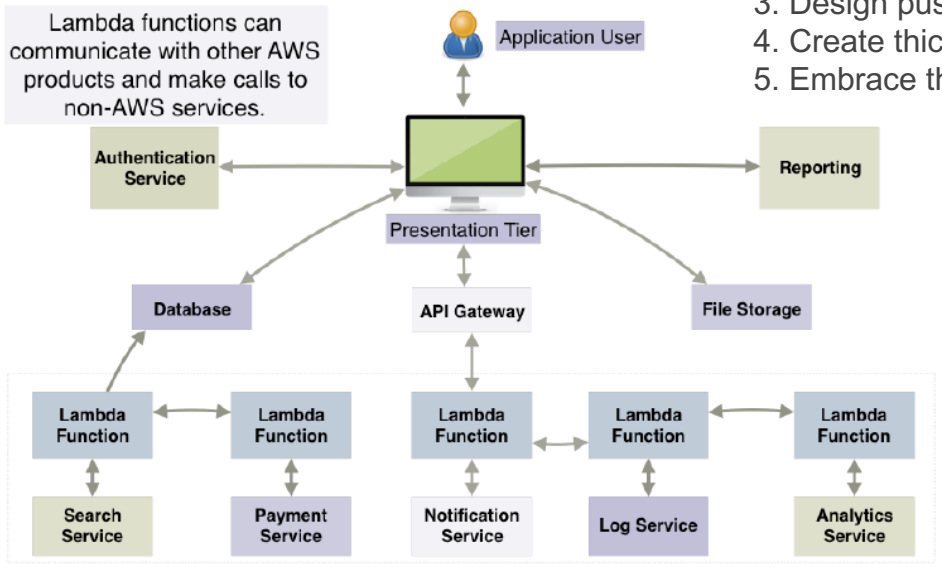


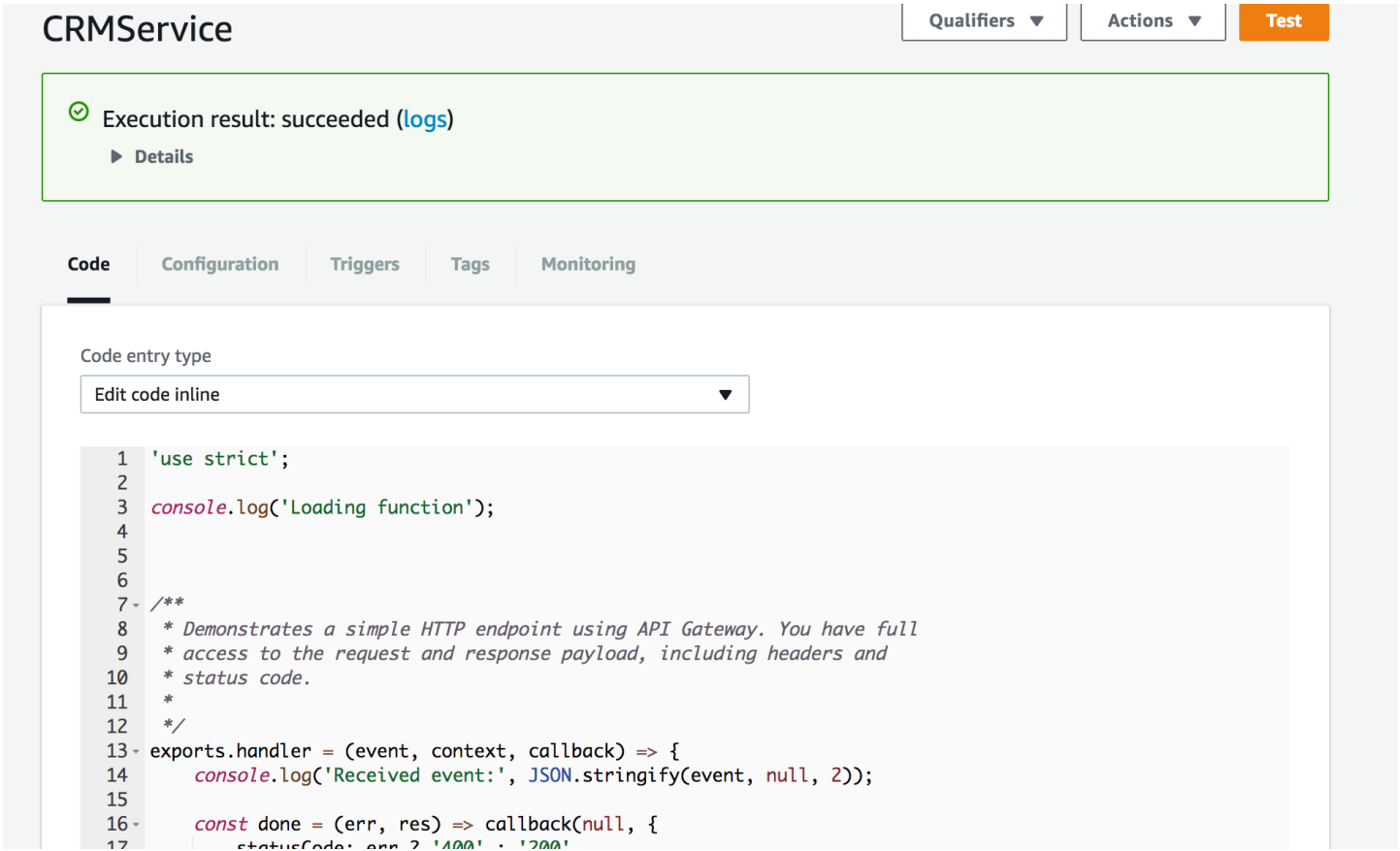
Figure 1.3: In a serverless architecture there is no single traditional back end. The front end of the application communicates directly with services, the database, or compute functions via an API gateway. Some services, however, must be hidden behind compute service functions where additional security measures and validation can take place.

Some observations on serverless

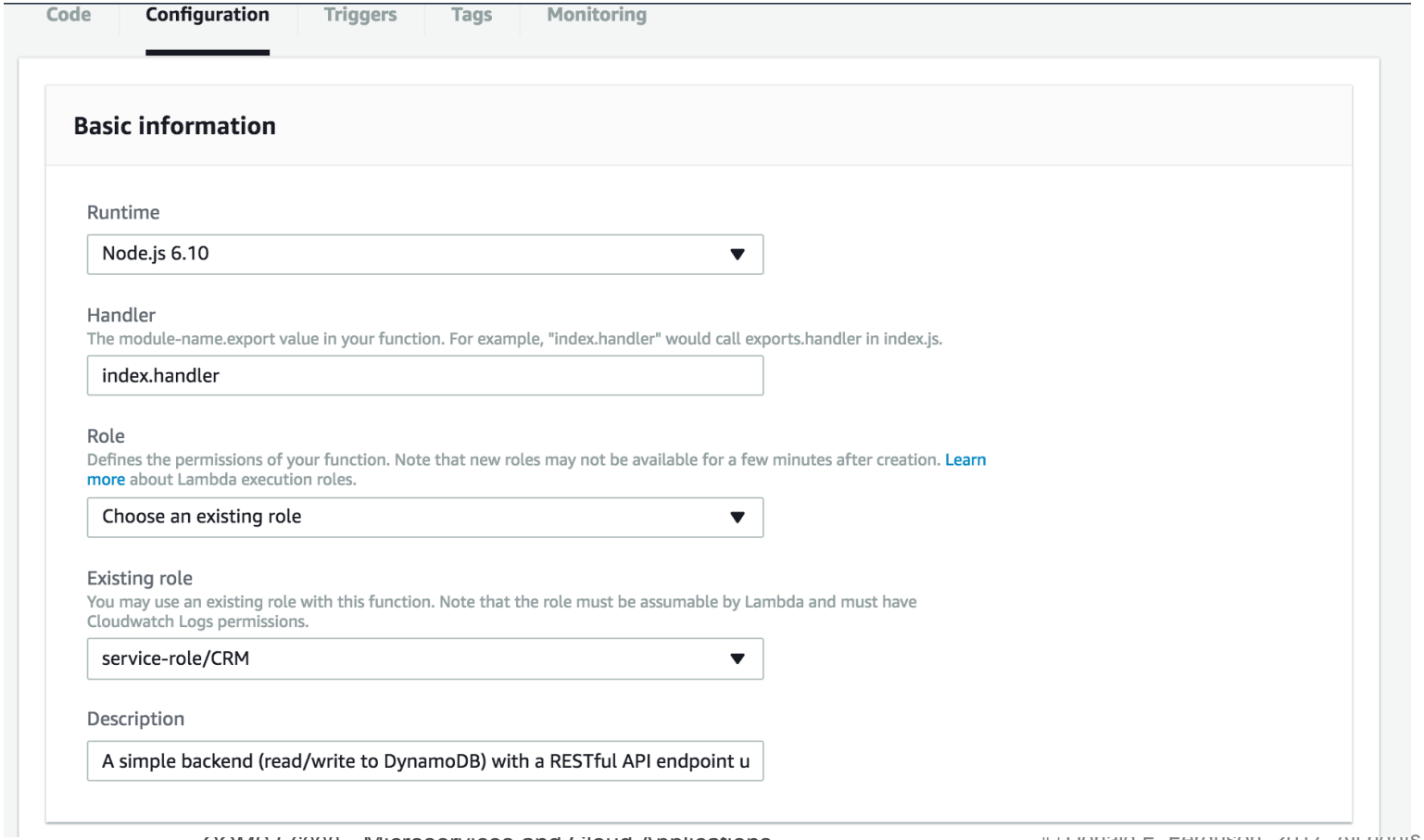
- There is running code → "some server somewhere."
- In IaaS,
  - You get the virtual sever from the cloud.
  - But know it is there, and manage and config it.
- In PaaS/microservices
  - You are aware of/build the "application sever."
  - And supporting frameworks.
  - And bundle/tarball it all together.
- In serverless,
  - You write a function based on a template.
  - Upload to an internet "event" endpoint.
  - Anything you call is a "cloud service."

- Let's build a Lambda function.
- Let's publish through the API Gateway

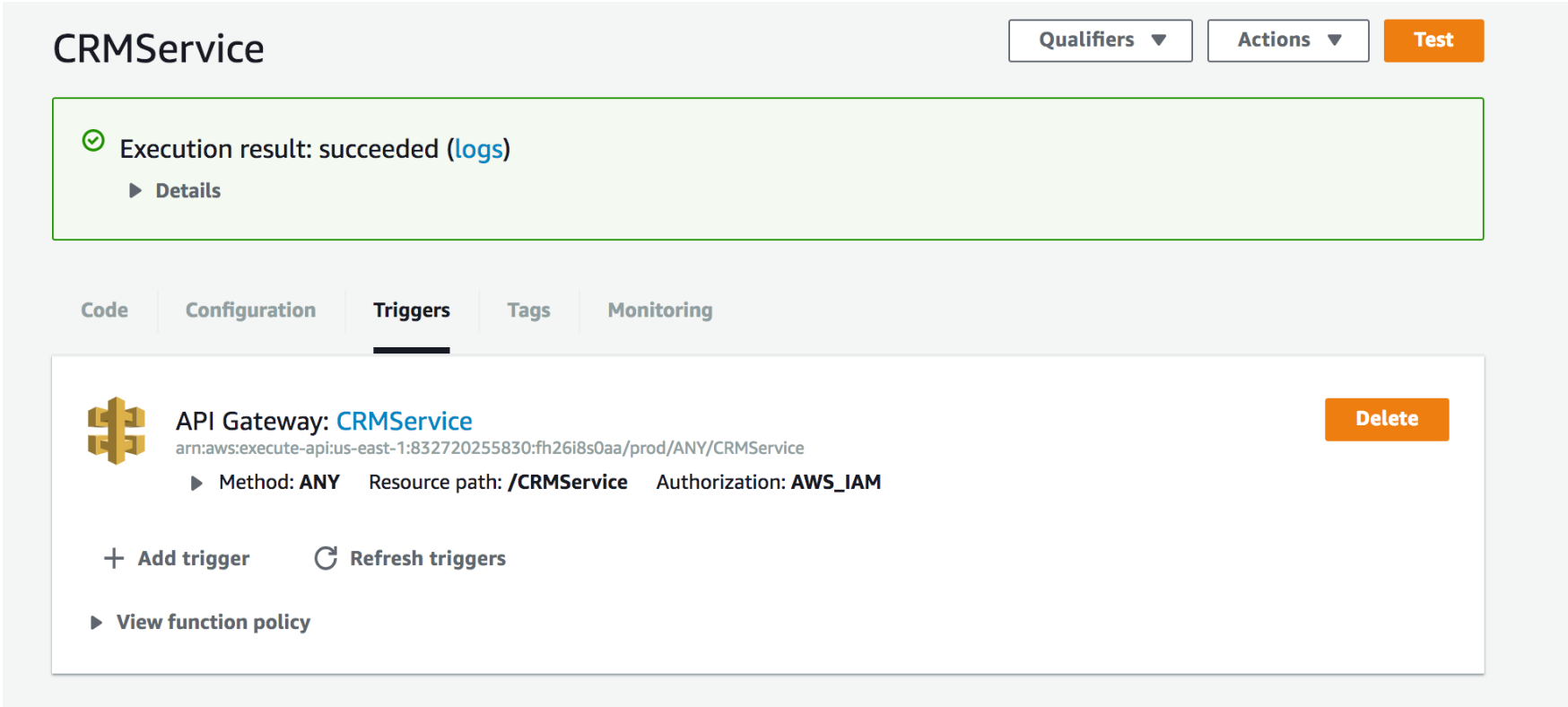
Lambda Function



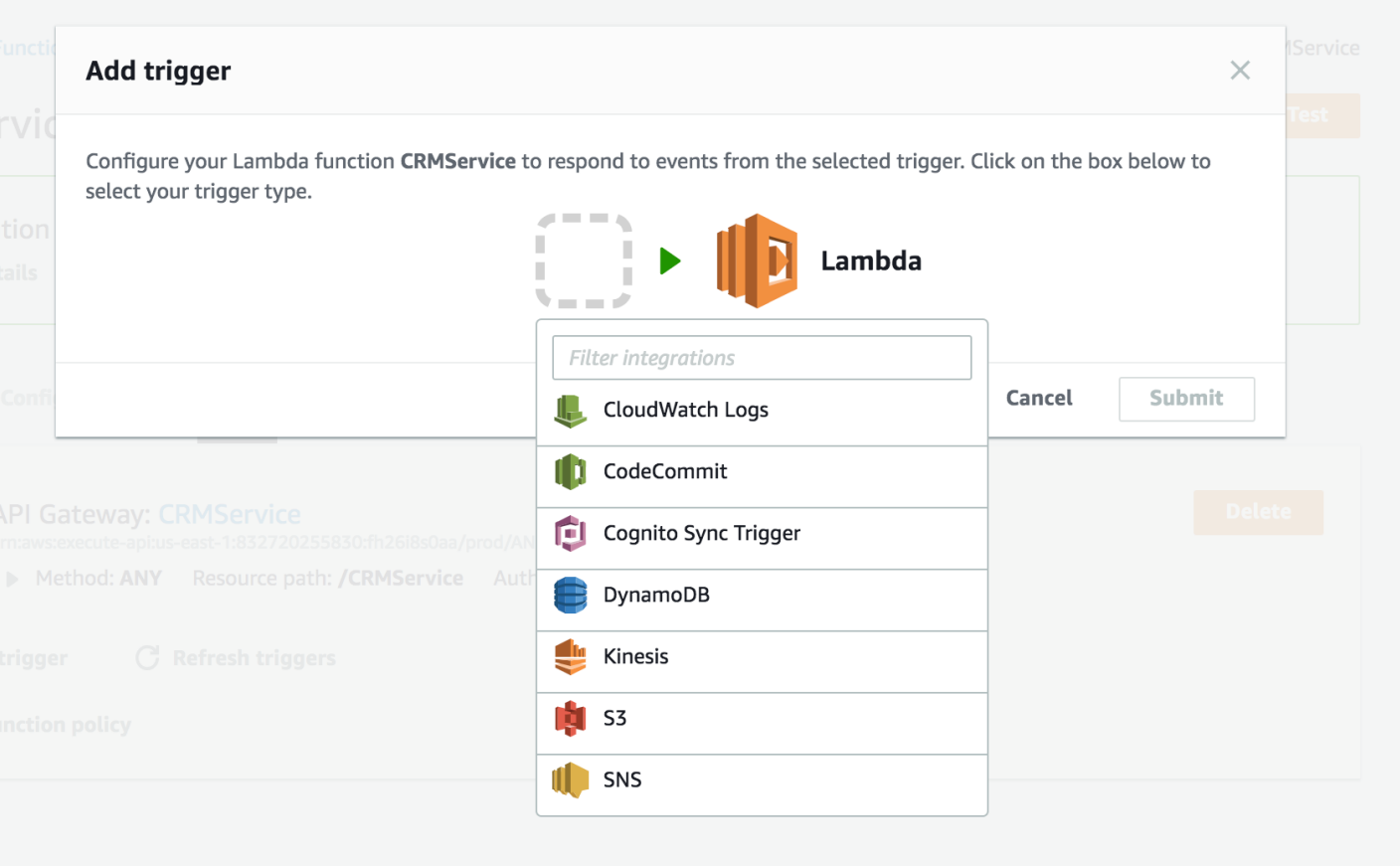
Lambda Function



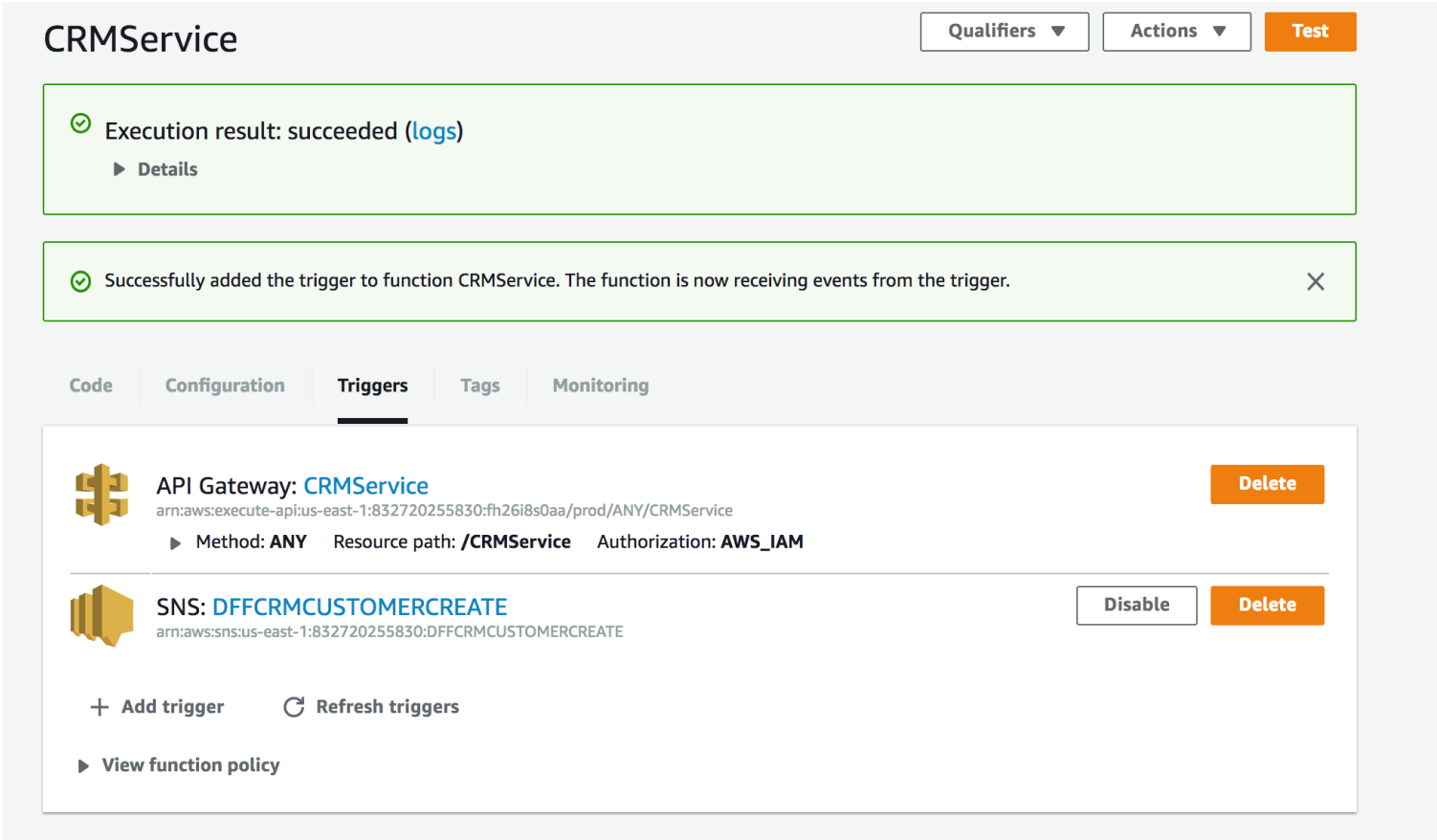
Lambda Function



Lambda Function



Lambda Function



Execution

Filter events

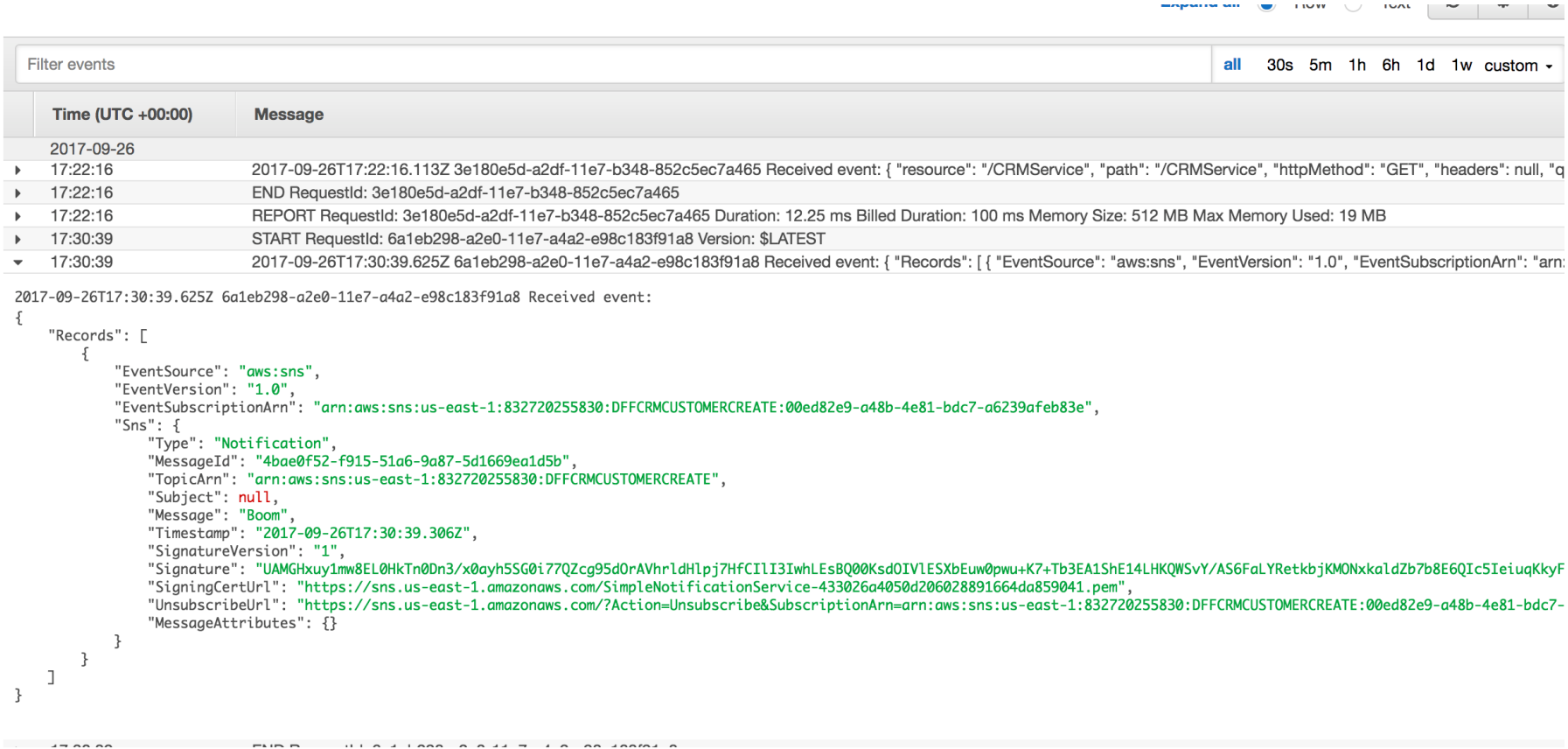
all 30s 5m 1h 6h 1d 1w custom

Time (UTC +00:00)	Message

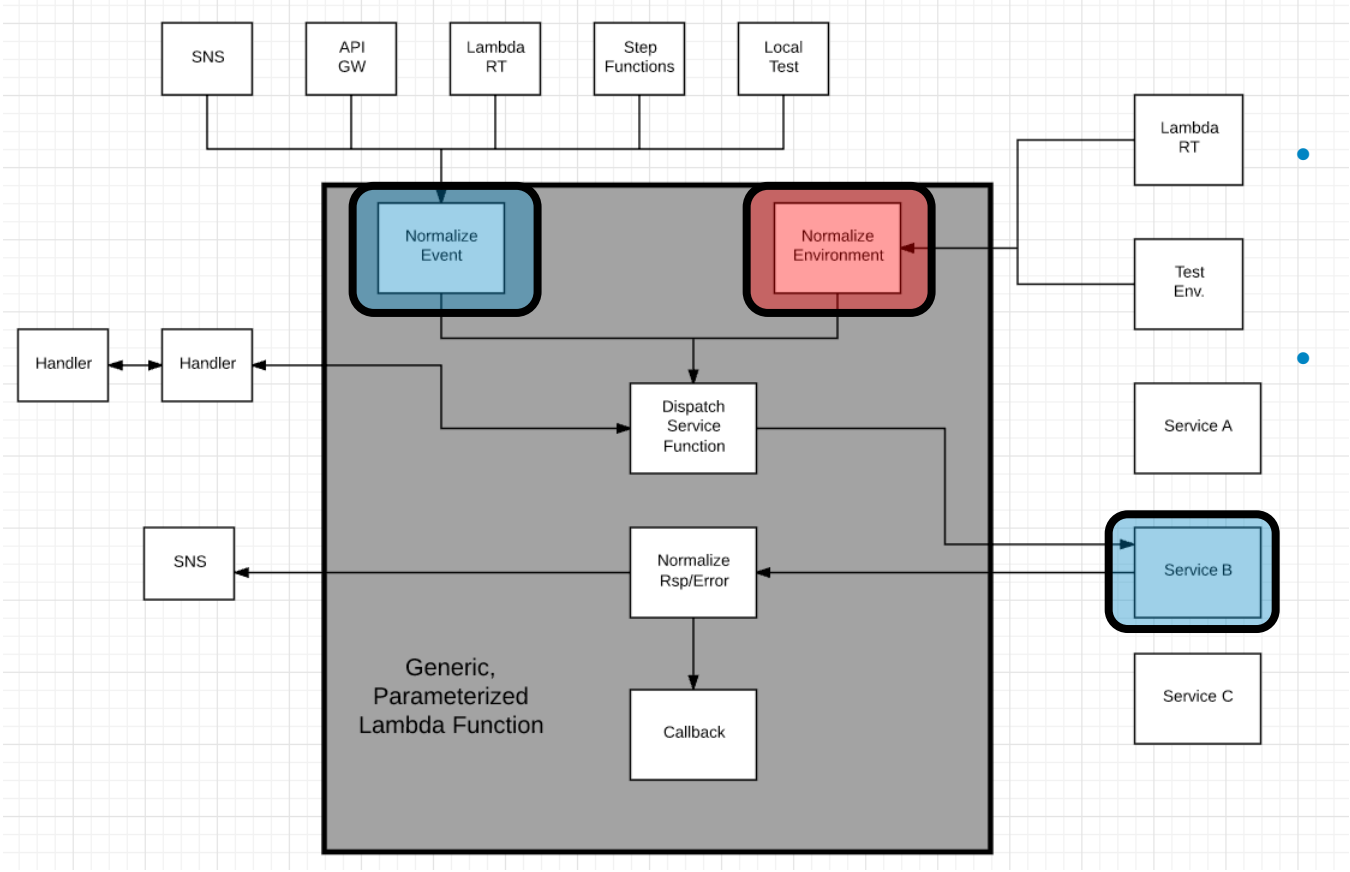
2017-09-26


```
{
    "resource": "/CRMService",
    "path": "/CRMService",
    "httpMethod": "GET",
    "headers": null,
    "queryStringParameters": null,
    "pathParameters": null,
    "stageVariables": null,
    "requestContext": {
        "path": "/CRMService",
        "accountId": "832720255830",
        "resourceId": "moj67a",
        "stage": "test-invoke-stage",
        "requestId": "test-invoke-request",
        "identity": {
            "cognitoIdentityPoolId": null,
            "accountId": "832720255830",
            "cognitoIdentityId": null,
            "caller": "832720255830",
            "apiKey": "test-invoke-api-key",
            "sourceIp": "test-invoke-source-ip",
            "accessKey": "ASIAIM6Y2A2G26GECZTQ",
            "cognitoAuthenticationType": null,
            "cognitoAuthenticationProvider": null,
            "userArn": "arn:aws:iam::832720255830:root",
            "userAgent": "Apache-HttpClient/4.5.x (Java/1.8.0_131)",
            "user": "832720255830"
        },
        "resourcePath": "/CRMService",
        "httpMethod": "GET",
        "apiId": "fh26i8s0aa"
    },
    "body": null,
```

Execution



Design Pattern – Generic Lambda Function



- Options
  - In zip file
  - Separate Lambda
- Config info
  - Added to Lambda
  - Env Variable
  - From S3

Generic Lambda Function

```
lambda_config.services  = {
    properties: require('./properties_svc').svc(),
    franchises: require('./franchises_svc').svc(),
    series: require('./series_svc').svc(),
    episodes: require('./episodes_svc').svc(),
    content_instances: require('./content_instances_svc').svc(),
    s3_assets:  require('./s3_assets_svc').svc(),
    related_assets:  require('./related_assets_svc').svc(),

};

lambda_config.path_mappings     =   {
    properties:     {
        paths:      ['/properties', '/properties/{id}'],
        service:    lambda_config.services.properties
    },
    franchises:     {
        paths:      ['/franchises', '/franchises/{id}'],
        service:    lambda_config.services.franchises
    },
    series:     {
        paths:      ['/series', '/series/{id_string}'],
        service:    lambda_config.services.series
    },
    episodes:       {
        paths:      ['/episodes', '/episodes/{id}'],
        service:    lambda_config.services.episodes
    },
    content_instances:      {
        paths:      ['/content_instances', '/content_instances/{id}'],
        service:    lambda_config.services.content_instances
    },
    s3_assets:      {
        paths:      ['/s3_assets', '/s3_assets/{id}'],
        service:    lambda_config.services.content_instances
    },
    related_assets:     {
        paths:      ['/related_assets', '/related_assets/{id}'],
        service:    lambda_config.services.related_assets
    }
};
```

- Multiple paths may map to the same Lambda function (or microservice), which dispatched onto sub modules.
- Load submodules from zip/deploy bundle. These may be
  - The *actual* implementation code.
  - A proxy that invokes the code deployed in a separate Lambda function.
  - A proxy that invokes the code via HTTP.
  - etc.
- Relies on
  - The module all having the same basic "shape."
  - Being binding (HTTP, local call, Lambda) independent.

Generic Lambda

```
if (service) {

    switch (normalized_event.verb) {

    case    'GET'   :
        if ((normalized_event.query_params) && (normalized_event.query_params.length > 0)) {
            get_by_query_params(service, normalized_event.query_params, context, normalize_callback);
        }
        else {
            get_by_id(service, normalized_event.keys, context, event, normalize_callback);
        }
        break;
    case    'PUT'   :
        update_by_id(service, normalized_event.keys, context, event, normalize_callback);
        break;
    case    'POST'  :
        create(service, normalized_event.data, context, event, normalize_callback);
        break;
    case    'DELETE'    :
        delete_by_id(service, normalized_event.keys, context, event, normalize_callback);
        break;
    case    'ECHO'  :
        echo(service, normalized_event, context, event, normalize_callback);
        break;
    default:
        generic_verb(service, normalized_event, context, event, normalize_callback);
        break;
    }
}
else {
    normalize_callback(errors.not_found, null);
}
```

- Echo
  - Test connectivity.
  - Heartbeat for monitoring
- Pass not “REST” verbs directly to service.

Configured Service

```
exports.svc     =   function() {

    var     base                =   generic_svc.svc(svc_config);

    var     module_name         =   svc_config.module_name;

    var     svc = {};

    svc.echo                    =   base.echo;
    svc.get_by_id               =   base.get_by_id;
    svc.get_by_query_params     =   base.get_by_query_params;
    svc.create                  =   base.create;
    svc.update_by_id            =   base.update_by_id;
    svc.delete                  =   undefined;

    init();

    function init() {
    }

    return svc;
};
```

In an extreme case, a specific service can just delegate onto a generic service and pass configuration information.

Configured Service

```
svc.create              =   function(object, context) {
    var function_name   =   config.module_name + '.create: ';
    logging.debug_message(function_name + "Enter. object =  ", object);

    return new Promise(function(resolve, reject) {

        object[config.id_property]  = uuid.v1();
        logging.debug_message(function_name + " UUID wil be " + object.id);

        if (config.required_create_fields.indexOf('created') !== -1) {
            object.created      =   new Date().getTime();
        }

        if (valid_create_fields(object)) {
            logging.debug_message(function_name + "create is valid");
            config.db.create(object).then(
                function(resource) {
                    resource    =   process_single_response(resource);
                    resolve(resource);
                },
                function(error) {
                    reject(error);
                });
        }
        else {
            reject(errors.invalid_create);
        }
    });
};
```

- Automatically set generated values, e.g.
  - Created, Modified
  - UUID
- Validate body
  - All required fields.
  - No extra fields.
  - Apply constraint functions.
- Process data returned from DB layer.
  - Return on fields that match definition.
  - Convert foreign keys into links.
- … …