

# Composer API 0.19.x

## Discount Coupon Links to UDEMY courses:



<https://www.udemy.com/hyperledger/?couponCode=DKHLF1099>



<https://www.udemy.com/ethereum-dapp/?couponCode=DKETH1099>



<https://www.udemy.com/rest-api/?couponCode=DKRST1099>



mentoring, seeking Blockchain part time work, project guidance, advice ... ..  
<http://www.bcmentors.com>

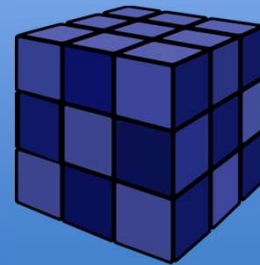
This deck is part of a online course on “Hyperledger Fabric Development with Composer”

raj@acloudfan.com



@acloudfan

<http://ACloudFan.com>



# Subscribing to Events

## Learning Objectives:

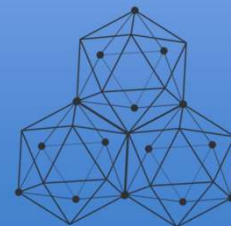
- Using Client API
- Using the REST Server

raj@acloudfan.com



@acloudfan

<http://ACloudFan.com>





## Incremental Creation of ACME Air Domain Model

<https://github.com/acloudfan/HLF-Course-Domain-Model>

<http://www.ACloudFan.com>

```
▸ airlinev1  
▸ airlinev2  
▸ airlinev3  
▸ airlinev4  
▸ airlinev5  
▸ airlinev6  
▸ airlinev7  
▸ airlinev8  
▸ airlinev9
```



Code shown in video may change over time



## Hyperledger Fabric API

<https://github.com/acloudfan/HLF-Fabric-API>

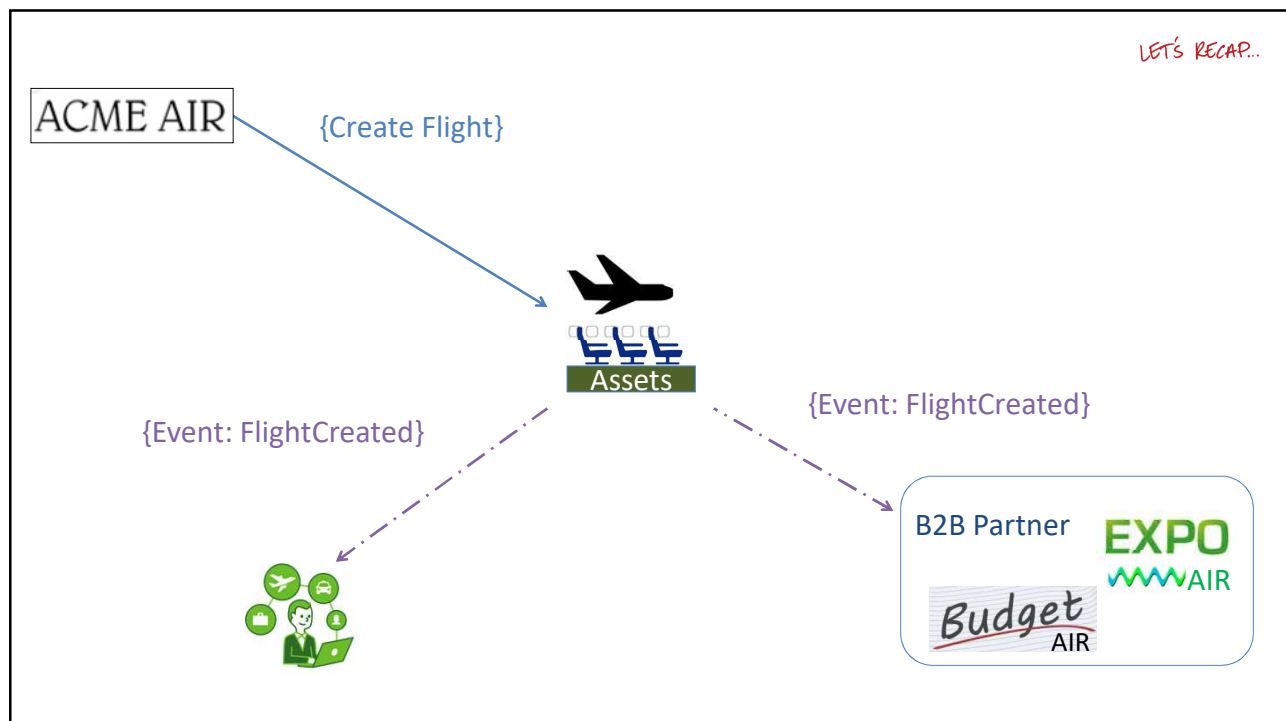
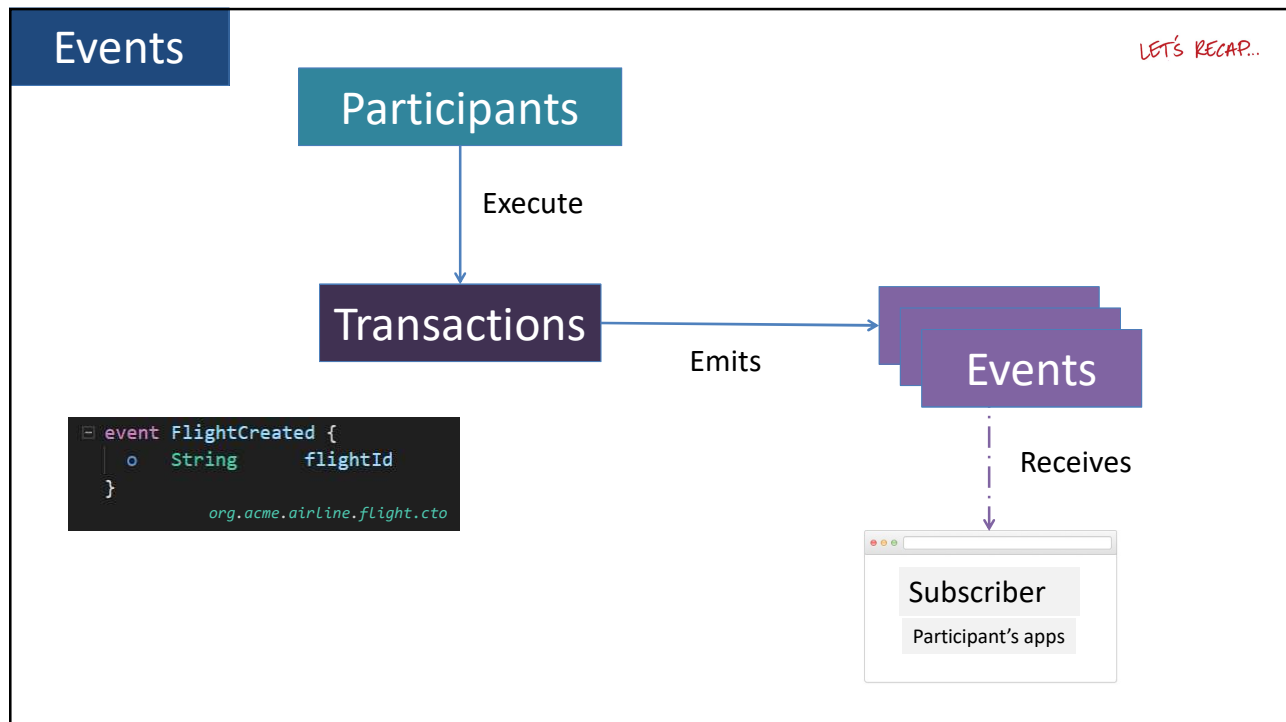
Samples demonstrating use of various Fabric SDK/API

<http://www.ACloudFan.com>

```
JS subscribe-event.js
```

```
JS subscribe-event-ws.js
```

Code shown in video may change over time



## Subscription

### Subscriber receives ALL events

- Criteria based event subscription **NOT** supported

*As of January 2018*



- Application logic needed for event filtering

## Events

### Two ways of event subscription



- Event subscription **API**



- **Websocket** connection to Composer **REST Server**

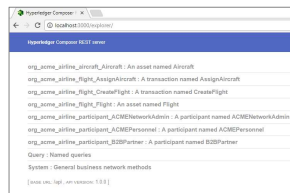
## Events

## Two ways of event subscription



## Events

## Testing the sample



## 2. Execute `CreateFlight` transaction

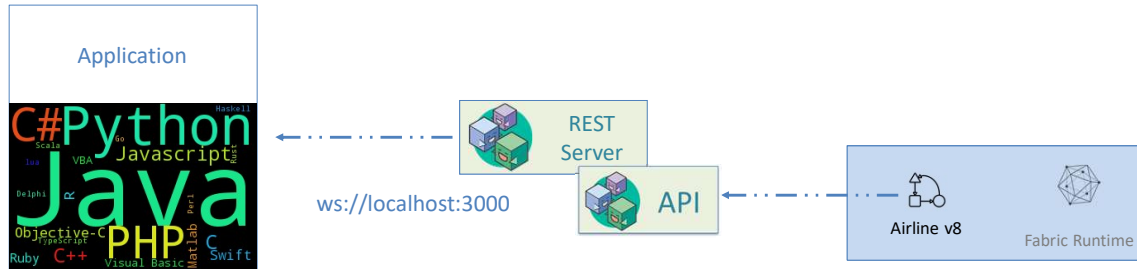
JS subscribe-event.js

### 1. Launch the subscriber sample

3. >>> Console message <<<

## Events

## REST Server WebSocket interface



## Events



- Available for Javascript | NodeJS
- Use it with any language that supports WS

## Units Testing of Network Apps

raj@acloudfan.com

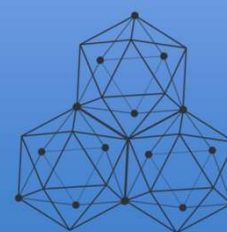
 @acloudfan

<http://ACloudFan.com>

### Learning Objectives:

<http://www.ACloudFan.com>

- Writing unit test cases



## Hyperledger Fabric API

<https://github.com/acloudfan/HLF-Fabric-API>

Samples demonstrating use of various Fabric SDK/API

<http://www.ACloudFan.com>



```
JS my-test.js
```

Code shown in video may change over time

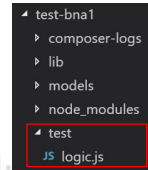
## UNIT TESTS

- Option#1

Use the test template generated by **yo**



> yo hyperledger-composer



- Option#2

Create the test case file by copying content from

JS mocha-ut-harness-template.js

## UNIT TESTS

STEP ①

Arrange



Deploy the BNA to Runtime & connect

STEP ②

Act



Setup code for the Test cases

STEP ③

Assert



Check for Pass | Fail

Execute



mocha test-file.js



# UNIT TESTS

STEP ①

Arrange



JS mocha-ut-harness-template.js

```
const utHarness = require('./ut-harness');
```

```
// This points to the model project folder
var modelFolder = 'Your-Project-Folder'

var adminConnection = {}
var businessNetworkConnection = {}
var bnDefinition = {}
```

- Copy to your project folder & rename
- Change path to the ut-harness.js
- Change the model folder path

ACloudFan.com

Synchronous call

```
// Synchronous call so that connections can be established
before((done) => {
  utHarness.debug = false;
  utHarness.initialize(modelFolder, (adminCon, bnCon, definition) => {
    adminConnection = adminCon;
    businessNetworkConnection = bnCon;
    bnDefinition = definition;
    done();
  });
});
```

Initializes

# UNIT TESTS

STEP ②

Act



- Use `describe()` for creating test suites
- Code the test cases within `it()` blocks
- Use the connection objects in your code

http://www.ACloudFan.com

```
// Test Suite # 1
describe('Give information on the test case', () => {

  // Test Case # 1
  it('should have more that 1 registry', () => {
    Code your test case
  });

  // Test Case # 2
  it('should have more that 2 registry', () => {
    Code your test case
  });
});
```

# UNIT TESTS

STEP ③

Assert



- Decide on **assertion** style

```
var assert = require('chai').assert;
```

<http://www>

```
// BDD style:
var expect = require('chai').expect;
// Or
var should = require('chai').should;
```

- Add the **assertions**

```
// Test Case # 1
it('should have more than 1 registry', () => {
  // Your test code goes here

  // expression in assert
  assert(true)
});
```

# UNIT TESTS

Execute

```
>_ mocha test-file.js
```

<http://www.ACloudFan.com>

## Runtime API Class



### Learning Objectives:

- Global functions for Transaction Processors
- Managing Assets | Participants

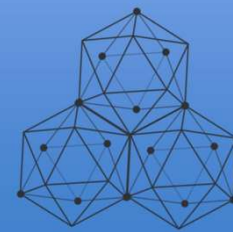


- Error processing

raj@acloudfan.com

@acloudfan

<http://ACloudFan.com>



## Incremental Creation of ACME Air Domain Model

<https://github.com/acloudfan/HLF-Course-Domain-Model>

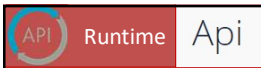
<http://www.ACloudFan.com>

```

▸ airlinev1
▸ airlinev2
▸ airlinev3
▸ airlinev4
▸ airlinev5
▸ airlinev6
▸ airlinev7
▸ airlinev8
▸ airlinev9
  
```



Code shown in video may change over time



- Class that contains **ROOT** of the **transaction processor API**



Fabric 1.x

Runtime

```

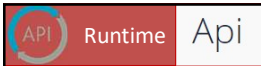
/**
 * Create Flight Transaction
 * @param {org.acme.airline.flight.CreateFlight} flightData
 * @transaction
 */
function createFlight(flightData) {
  }

```



Runtime

Api

Functions available  
by default

Transaction processor code manages the resource instance



Fabric 1.x

Runtime

getAssetRegistry( ... )



Runtime

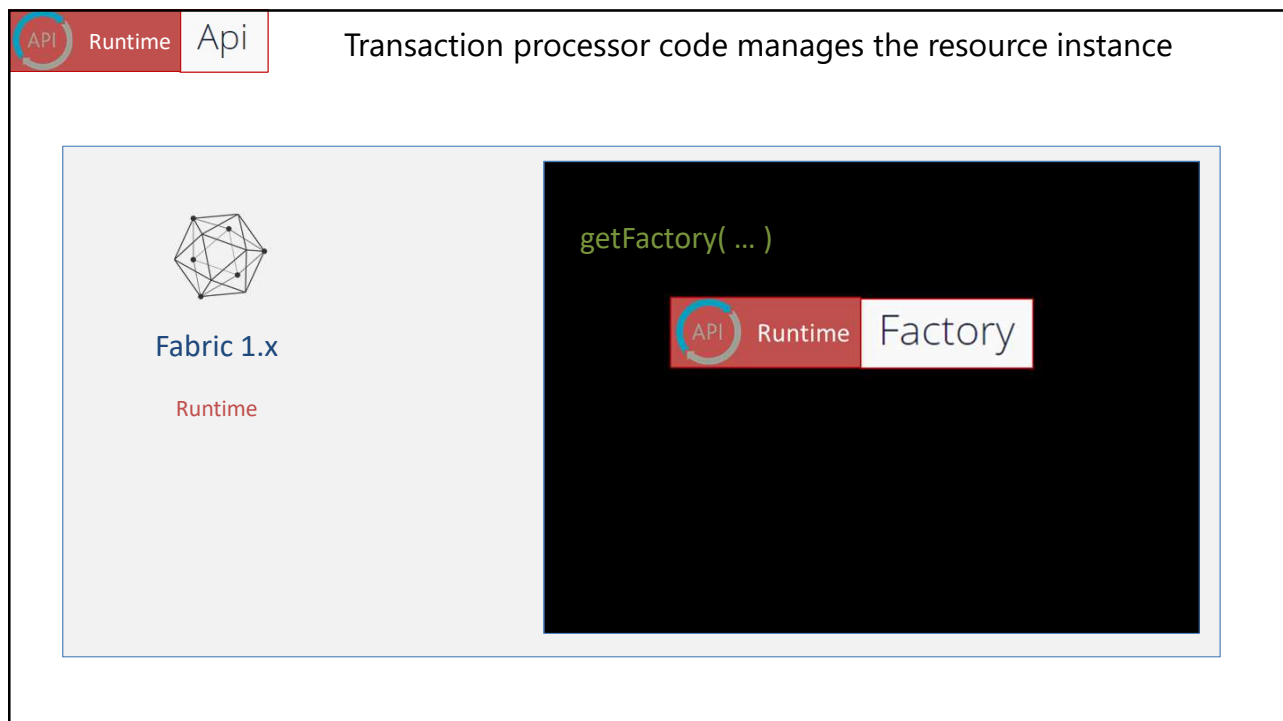
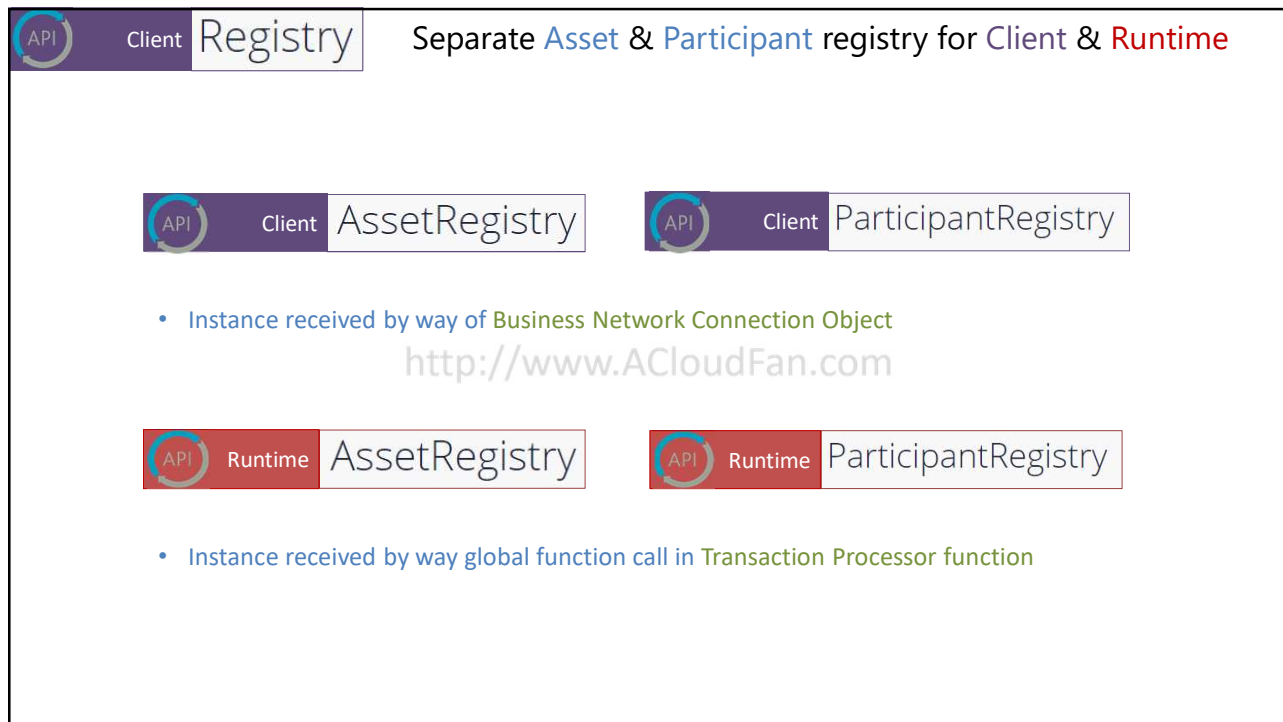
AssetRegistry

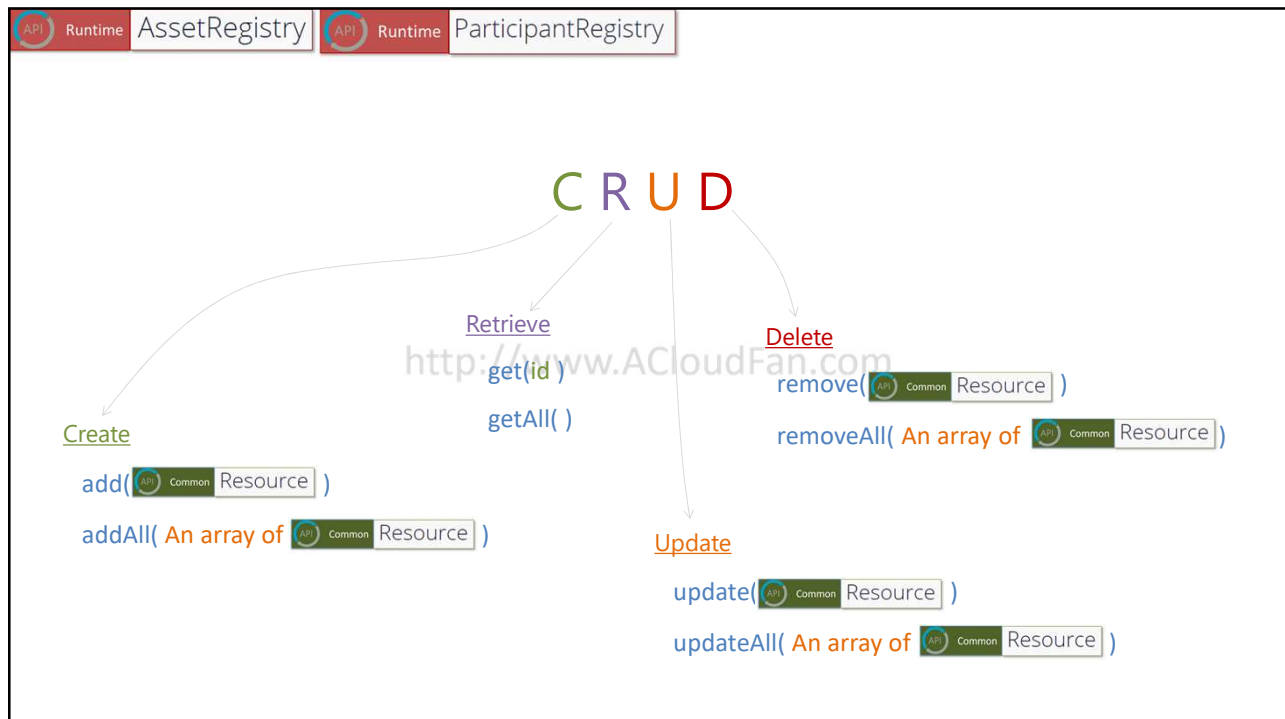
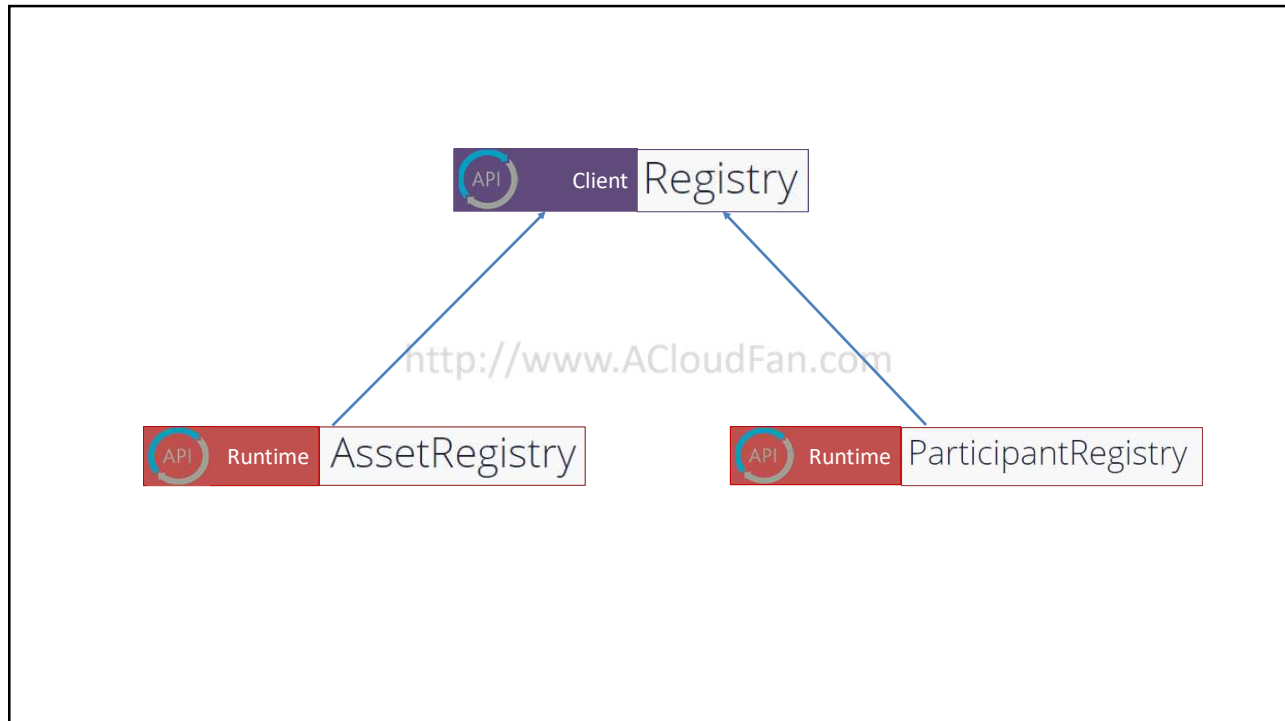
getParticipantRegistry( ... )



Runtime

ParticipantRegistry





API

Runtime

Api

## Global functions

```

/**
 * Create Flight Transaction
 * @param {org.acme.airline.flight.CreateFlight} flightData
 * @transaction
 */
function createFlight(flightData) {
  ... ..
  return getAssetRegistry('org.acme.airline.flight.Flight')
  var factory = getFactory();
  ... ..
}

```

oudFan.com

No need for BusinessNetworkConnection


API

Runtime

Api

## Throw an error

- Rolls back all changes prior to the throw



Fabric 1.x  
Runtime

```

/**
 * Create Flight Transaction
 * @param {org.acme.airline.flight.CreateFlight} flightData
 * @transaction
 */
function createFlight(flightData) {
  ... ..
  if ( some-condition ) {
    // Rollback all changes & inform caller about the error
    throw new Error('Provide an appropriate message ....');
  }
  ... ..
}

```



```
airlinev8
function createFlight(flightData)
```

1. Throw an error if **schedule date is past**
2. Add a **Flight** asset

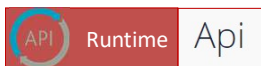
<http://www.ACloudFan.com>

```
airlinev8
JS schedule-validation.js
```

3. Create a Unit Test case file

```
it('should create & retrieve 1 Flight instance')
it('should throw exception if schedule is past date')
```

## Programmatic Access Control



### Learning Objectives:

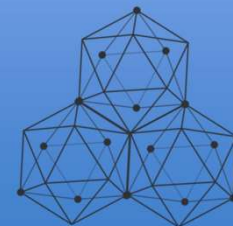
- Coding the access control

<http://www.ACloudFan.com>


raj@acloudfan.com

 @acloudfan

<http://ACloudFan.com>



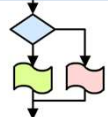





Access Control

## Access Control Implementation

LET'S RECAP...



Programmatic



Declarative


Coded

 in the transaction processing functions

- Based on the user context & transaction data

Rules defined *Access Control Language*

<http://www.ACloudFan.com>



Runtime


Api



Fabric 1.x

Runtime

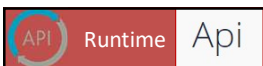
getCurrentParticipant( )



Common

Resource

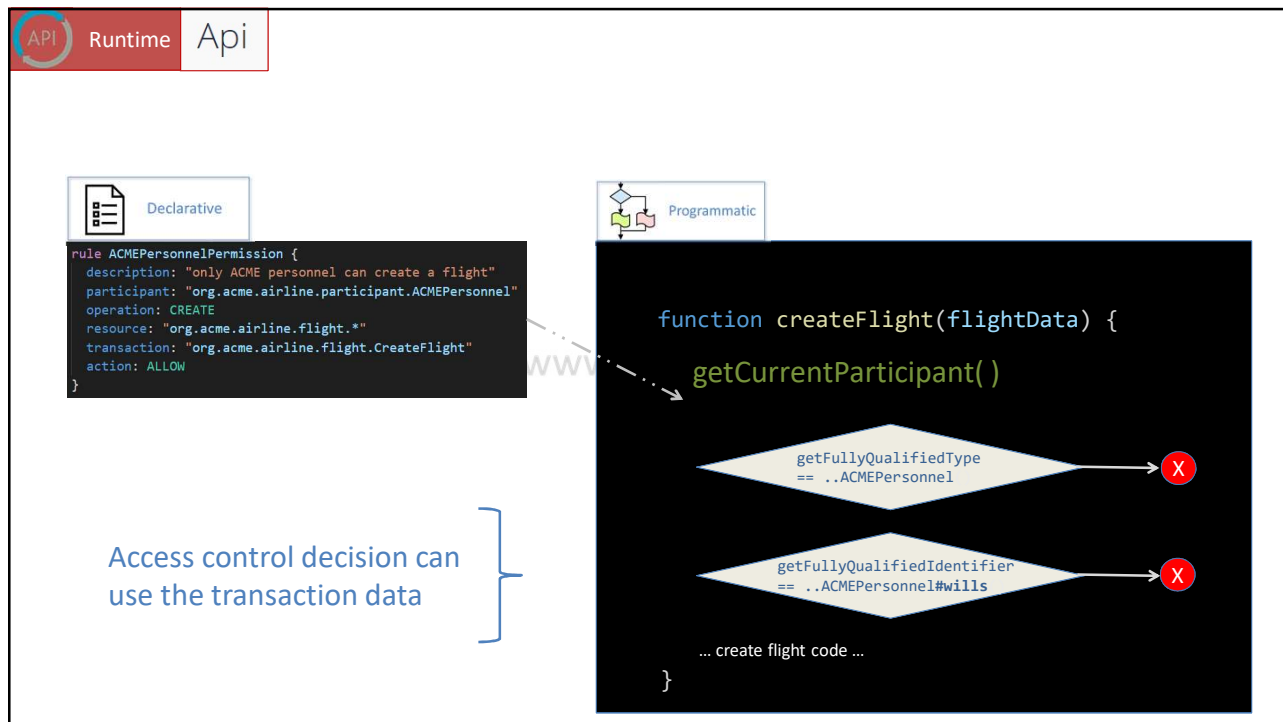
- Null     If participant identity was not used



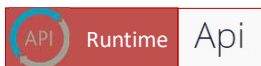
```
rule ACMEPersonnelPermission {
  description: "only ACME personnel can create a flight"
  participant: "org.acme.airline.participant.ACMEPersonnel"
  operation: CREATE
  resource: "org.acme.airline.flight.*"
  transaction: "org.acme.airline.flight.CreateFlight"
  action: ALLOW
}
```



}



## Queries (Transaction Processor)



### Learning Objectives:

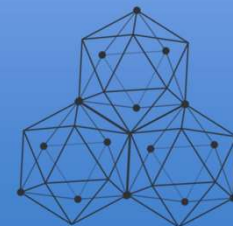
- Support for querying
- Usage scenarios

<http://www.ACloudFan.com>

raj@acloudfan.com

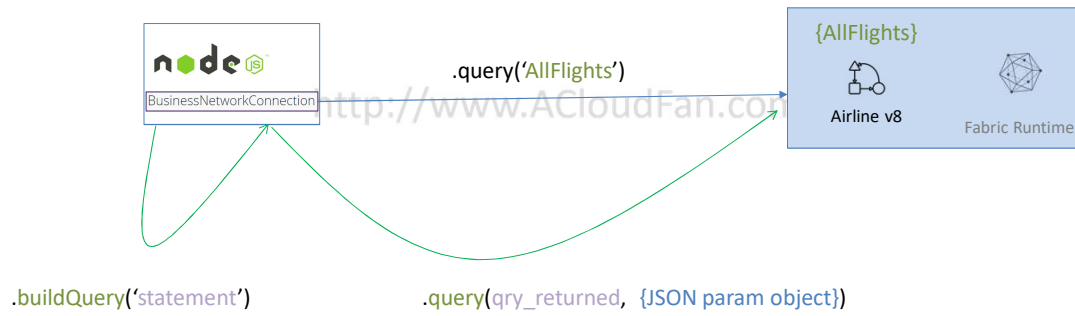
@acloudfan

<http://ACloudFan.com>



## Client Queries

LET'S RECAP...



API Runtime Api



Fabric 1.x

Runtime

For world state Fabric setup needs to be:



`query(qry, param )`

- Execute the named query

`queries.qry`


- Execute query created with

`buildQuery( )`

`buildQuery(statement )`


- Creates a query for later execution

API Runtime Api



Fabric 1.x  
Runtime

For world state Fabric setup needs to be:



```
query(qry, param )
```

- Execute the named query queries.qry
- Execute query created with buildQuery( )

```
buildQuery(statement)
```

- Creates a query for later execution


Why would you need *Query* invocations?

<http://www.ACloudFan.com>

- Batch updates
- Calculations
- Consolidations | Reporting


## Exercise

- ACME aircrafts sometime go for unplanned maintenance
- All flights that are assigned the aircraft need to be updated with a new aircraftId/assignment
- Create a Transaction that would allow the Logistics team to do batch updates
- You may use a Named or Dynamic Query



Fabric 1.x  
Runtime

For world state Fabric setup needs to be:



```
UpdateAircraftInBatch( ... )
```

1. Query the Flights from today for N days that are assigned a specific aircraft
2. Update the assigned aircraft to passed aircraft ID

airlinev8 project

# Composer API 0.19.x

## Discount Coupon Links to UDEMY courses:



<https://www.udemy.com/hyperledger/?couponCode=DKHLF1099>



<https://www.udemy.com/ethereum-dapp/?couponCode=DKETH1099>



<https://www.udemy.com/rest-api/?couponCode=DKRST1099>



mentoring, seeking Blockchain part time work, project guidance, advice ... ..  
<http://www.bcmentors.com>

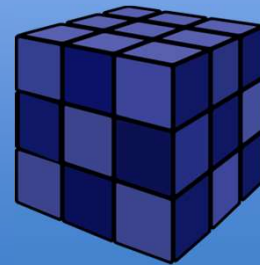
This deck is part of a online course on “Hyperledger Fabric Development with Composer”

raj@acloudfan.com



@acloudfan

<http://ACloudFan.com>



## Emitting events & Integrating with systems



### Learning Objectives:

- `emit()`
- `getSerializer()`
- Integrating with external systems using `post()`

raj@acloudfan.com



@acloudfan

<http://ACloudFan.com>

