Underrated

Nashorn Javascript

Presented by Stephen Russett
@stephenrussett
Github: StephenOTT

Underrated:

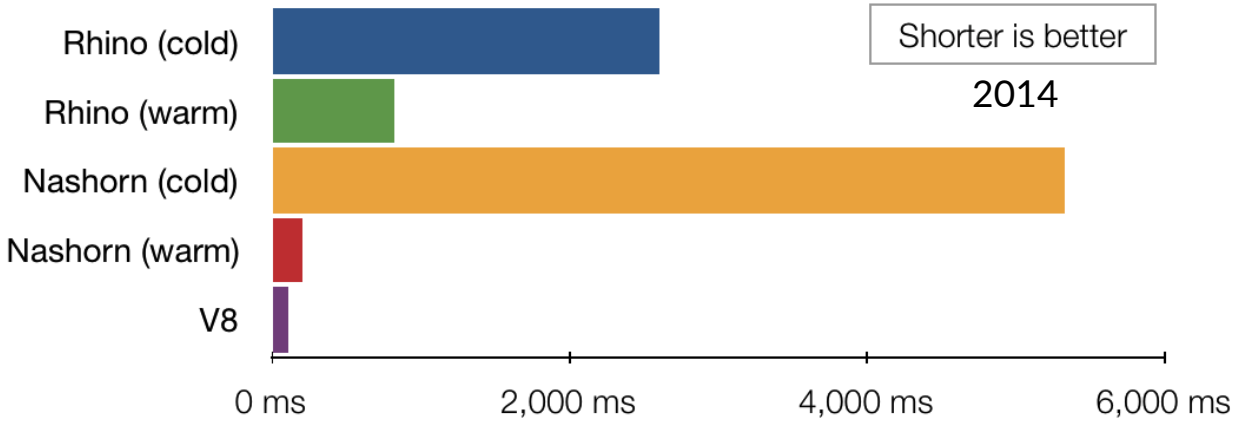
1. Nashorn Javascript Engine
2. Add JS to your Java apps
3. Using JS and Java…
4. Polyglot Vert.x
5. Project Es4x and Node/NPM
6. Automation/BPM and JS
7. Enterprises, Government, and all those difficult app environments

Wrong?: Execute Javascript on the JVM

Nashorn Javascript Engine

Nashorn introduced in Java 8 to replace Rhino engine

(Nashorn in German means Rhino)



Use as a command line scripting interface or **embed it**

Javascript -> bytecode -> JVM

My point:

not to replace V8 / node.

leverage the JVM capabilities if and when required, **and** still get to use scripting

Yes.. It's not "as fast as V8"!

Netflix Nashorn use case from 2015:

Dynamic Report Editing

https://www.infoq.com/news/2015/01/nashorn-at-netflix

Check out the following for more benchmarks:
https://lemnik.wordpress.com/2017/08/07/is-nashorn-jvm-faster-than-node-v8/

Add JS to your Java App

```
1. ScriptEngine nashornEngine = new ScriptEngineManager().getEngineByName("nashorn");
2.
3. String jsScript = inputStreamAsString(someJsFile);
4.
5. Bindings bindings = nashornEngine.createBindings();
6. bindings.put("personInstance", personClass.getBioData());
7.
8. Object scriptResult = nashornEngine.eval(jsScript, bindings);
```

Done

Some Additional Nashorn features


```
1.
2.  // import lodash from CDN
3.  load('https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.17.10/lodash.min.js')
4.
5.  // Print to stdout
6.  var System  = Java.type("java.lang.System")
7.  System.out.println('this prints to stdout')
8.
9.  // or just
10. print('this prints to stdout')
11.
12. // Working with JSON and returning something Java can deal with:
13. // myJson, if returned to the Java side would be available as a Java Map
14. var myJson = Java.asJSONCompatible({
15.   x: 343,
16.   y: 'hello',
17.   z: [ 2, 4, 5]
18.   });
19.
20. // Java Objects and Arrays, Maps and Collections
21. // are traversable as regular JSON in JS
22. var myValue = someJavaMap['someKey']
23.
24. // Pre es6 support, there was extra loop helpers:
25. for each (i in someArray) {
26.    print(i);
27. }
```

Little more full example:

```
1.
2.  // Create the Nashorn ScriptEngine
3.  ScriptEngine nashornEngine = new ScriptEngineManager().getEngineByName("nashorn");
4.
5.  // Get your script from a file
6.  String jsScript = inputStreamAsString(resource);
7.
8.  // Expose/Bind Java variables/objects into the Javascript:
9.  Bindings bindings = nashornEngine.createBindings();
10. bindings.put("someVariable", someVariable);
11. bindings.put("someVariable2", someClass.getSomeObject());
12.
13. // eval the script and get the specific result/what was returned by the engine
14. Object scriptResult = nashornEngine.eval(jsScript, bindings);
15.
16. // Or just eval and not worry about the return
17. nashornEngine.eval(jsScript, bindings);
18.
19. // Or can can call specific functions:
20. String functionName = 'someFunctionName'
21. CompiledScript nashornCompiled = ((Compilable) nashornEngine).compile(jsScript);
22. ((Invocable) nashornEngine).invokeFunction(functionName); // Supports arguments as well
```

Node, ES6, Detroit

Nashorn is **not** 100% ES6 or Node library compatible.

Lots of history on integrations and compatibility discussions.

Look it up if interested: Project Avatar, etc

Diminishing returns as more libs become incompatible with Nashorn.

Java 9 adds ES6 features: symbols, let, const, iterator, collections Map and Set, Templates, Binary and octal numbers

OpenJDK Project Detroit aims to add V8 into JVM as a “ScriptEngine”…

but it just recently started…

The answer is not always: “replace it with Node”

Freemarker Example with Import Scopes

```
1.  /**
2.   * Evaluate/Render a FreeMarker template
3.   *
4.   * @param string content The string content of a FreeMarker template.
5.   * @param string object The KeyValue object/JSON object for placeholder bindings.
6.   * @return string The rendered FreeMarker template.
7.   */
8.  function renderFreeMarkerTemplate(content, placeholderValues)
9.  {
10.
11.   var ScriptEngine = new JavaImporter(javax.script);
12.
13.   with (ScriptEngine) {
14.     var manager = new ScriptEngineManager();
15.     var engine = manager.getEngineByName('freemarker');
16.
17.     var bindings = engine.createBindings();
18.     bindings.put('placeholders', placeholderValues);
19.
20.     var rendered = engine.eval(content, bindings);
21.
22.     return rendered;
23.   }
24. }
25.
26. var placeholderValues = {
27.    "firstName": "John",
```

```
5.   * @param string object The KeyValue object/JSON object for placeholder bindings.
6.   * @return string The rendered FreeMarker template.
7.   */
8.  function renderFreeMarkerTemplate(content, placeholderValues)
9.  {
10.
11.   var ScriptEngine = new JavaImporter(javax.script);
12.
13.   with (ScriptEngine) {
14.     var manager = new ScriptEngineManager();
15.     var engine = manager.getEngineByName('freemarker');
16.
17.     var bindings = engine.createBindings();
18.     bindings.put('placeholders', placeholderValues);
19.
20.     var rendered = engine.eval(content, bindings);
21.
22.     return rendered;
23.   }
24. }
25.
26. var placeholderValues = {
27.    "firstName": "John",
28.    "lastName": "Smith"
29. }
30.
31. var renderedTemplate = renderFreeMarkerTemplate(content, placeholderValues);
32. renderedTemplate.toString();
33.
```

Polyglot Vert.x

A Eclipse project

VERT.X

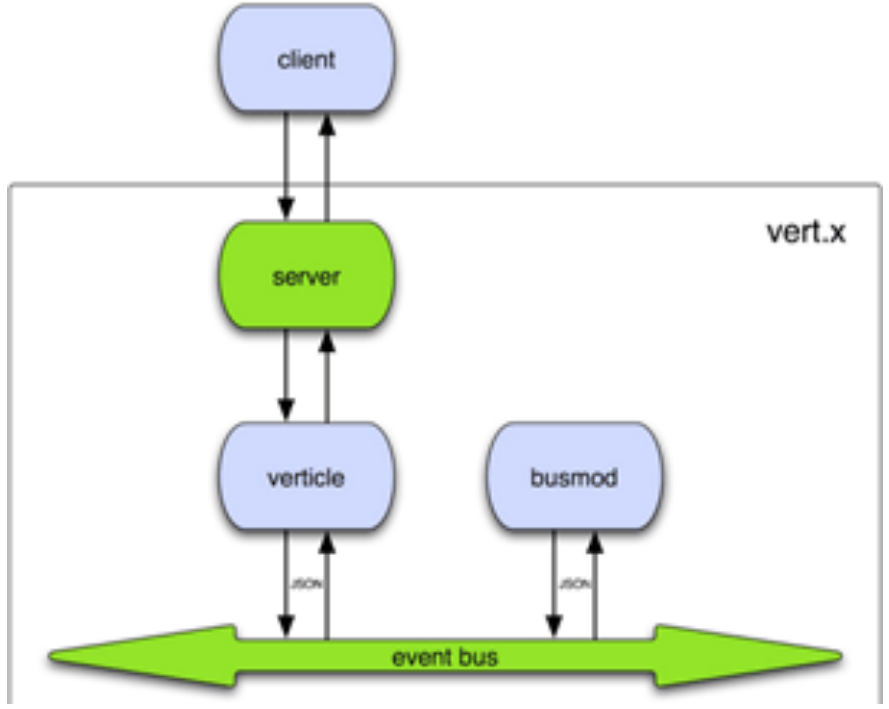
Vertx is a polyglot-reactive runtime that runs on the JVM.

They are telling me we can take this scripting even further

Build interoperable code on the JVM that will execute:

1. Java
2. Javascript
3. Groovy
4. Ruby
5. Ceylon
6. Kotlin
7. Scala

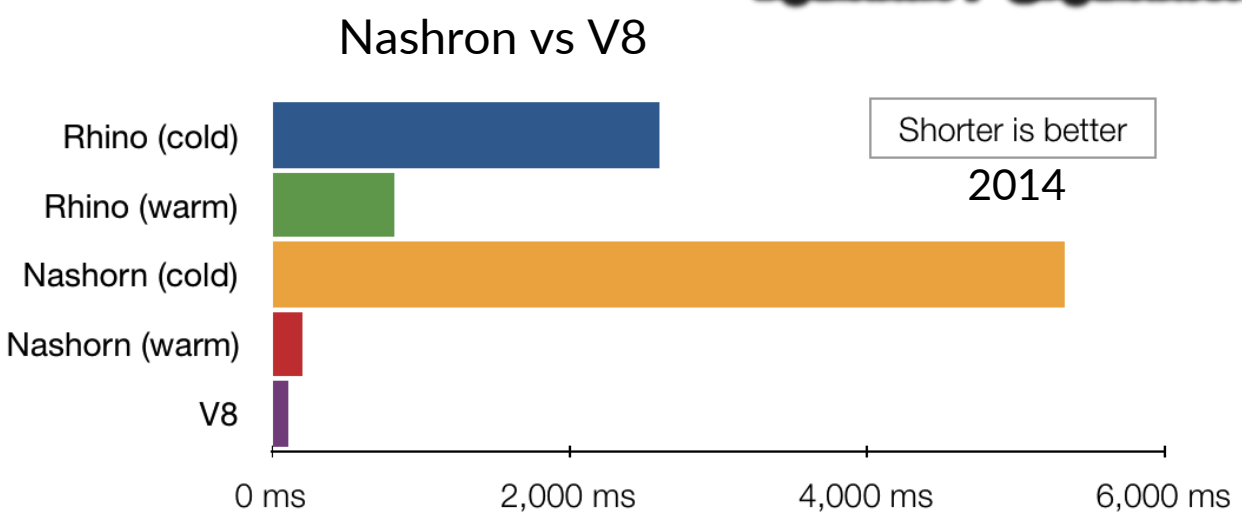
Btw.. It adds addition NPM and Node support on top of Nashorn



Look.. It can perform for most use cases...

But with obvious asterisks

And again, we are not trying to replace Node...



Vertx Javascript (Nashorn)* vs Node* ->

Deploy Multiple Languages within the app:

```
function deployVerticles() {
    vertx.deployVerticle('verticles/verticle1.js')
    vertx.deployVerticle('verticles/verticle2.gvy')
    vertx.deployVerticle('verticles/someWorker1.rb')
}
```

https://www.cubrid.org/blog/inside-vertx-comparison-with-nodejs

Nashorn + Java Drools + Vertx Web Service

```
1. // get a reference from Java to the JavaScript runtime
2. const DroolsHelper = Java.type('drools.DroolsHelper');
3. // get a drools engine instance
4. const engine = DroolsHelper.load(vertx.fileSystem().readFileBlocking("drools/rules.drl"));
5.
6. app.get('/greetings').handler(function (ctx) {
7.   // create a greetings message
8.   var greeting = DroolsHelper.createGreeting('Hello World!', function () {
9.     // when a match happens you should see this message
10.    console.log('Greetings from Drools!');
11.  });
12.
13.  // run the engine
14.  engine.insert(greeting);
15.  engine.fireAllRules();
16.
17.  // complete the HTTP response
18.  ctx.response().end();
19. });
20.
```

Vertx + es4x + NPM


```
// https://github.com/reactiverse/es4x
// terminal:
mkdir my-app
cd my-app
npm init -y
npm add vertx-scripts --save-dev
npm add @vertx/unit --save-dev
npm add @vertx/core --save-prod

// package.json:

{
  ...
  "scripts": {
    "postinstall": "vertx-scripts init",
    "test": "vertx-scripts launcher test -v",
    "start": "vertx-scripts launcher run",
    "package": "vertx-scripts package"
  },
  ...
}

// index.js:

/// <reference types="@vertx/core/runtime" />
// @ts-check
```

```
22. // index.js:
23.
24. /// <reference types="@vertx/core/runtime" />
25. // @ts-check
26.
27. vertx
28.   .createHttpServer()
29.   .requestHandler(function (req) {
30.     req.response().end("Hello!");
31.   })
32.   .listen(8080);
33.
34. console.log('Server listening at: http://localhost:8080/');
35.
36. // index.test.js:
37.
38. import { TestSuite } from '@vertx/unit';
39.
40. const suite = TestSuite.create("the_test_suite");
41.
42. suite.test("my_test_case", function (context) {
43.   var s = "value";
44.   context.assertEquals("value", s);
45. });
46.
47. suite.run();
48.
```

Start and Package your Node/NPM

`npm start` -> Run the App

Starts the JVM and executes the Vertx application

`npm package` -> builds a executable Jar that can be executed on your JVM

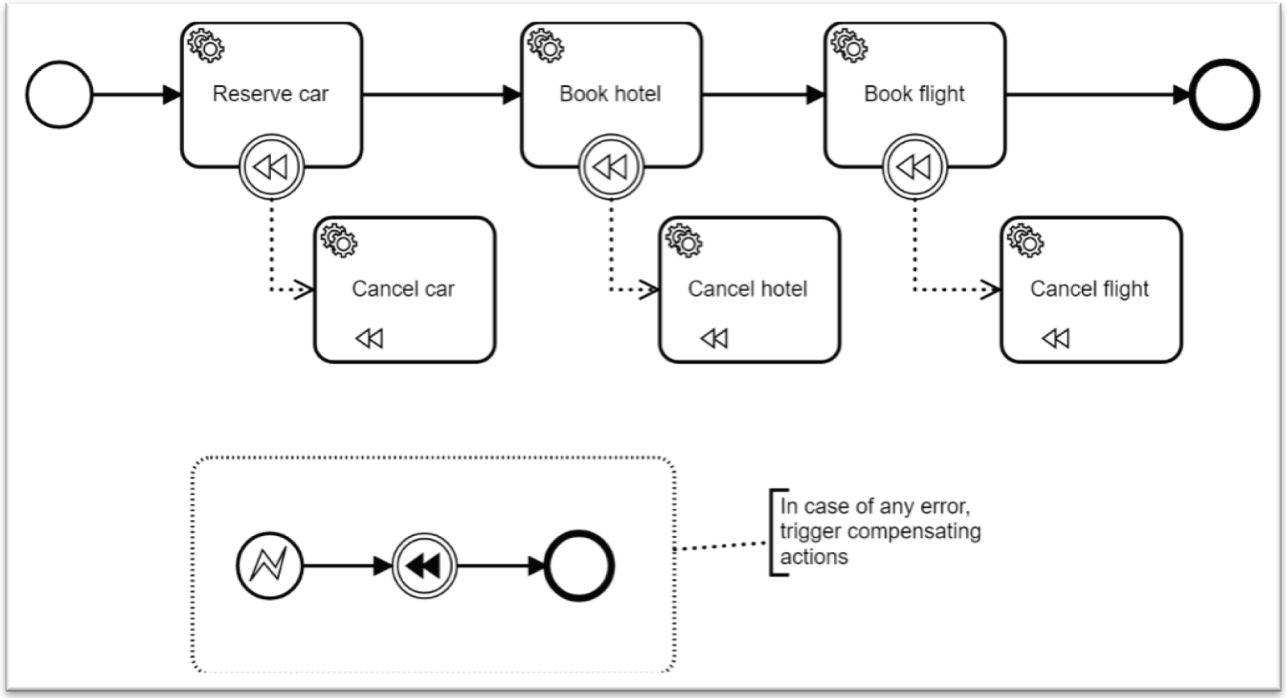
The “Magic” is it uses NPM as your development interface, but under the hood, its executing a Java build

https://github.com/reactiverse/es4x

Lets put it all together: Java BPMs and Javascript

Process Automation: Java dominated, not really any OSS non-Java contenders

More and more used for Microservices Orchestration:



Hmmmm, not sure this is for me…

Me: I don’t want to write JARs and “Java” for stuff… My technical business staff need to be able to write scripts!!

The Devs: I don’t want to write all of the spaghetti stupid business logic for every customer/client…

Use Case Examples!

1. Data Encryption
2. API Form Validations Scripting
3. Unit Testing / Writing Tests as Script
4. Incident Handlers
5. Extend the Camunda API with Vertx: Camunda Plugin and SpringBoot
6. BPMN Documentation Generator: Camunda BPMN Model Library, Freemarker and Vertx

Data Encryption

Encrypt / Seal

```
load('classpath:rsaEncrypt.js')
var messageToSeal = 'This message is to be hidden'
var sealedObject = rsaSeal(messageToSeal, PUBLIC_KEY)
execution.setVariable('sealed_object', sealedObject)
```

rsaEncrypt.js is a set of functions that call Java encryption classes

Decrypt / Unseal

```
load('classpath:rsaEncrypt.js’)
var sealedObject = execution.getVariable('sealed_object’)
var unsealedObject = rsaUnseal(sealedObject, PRIVATE_KEY)
execution.setVariable('unsealed_object', unsealedObject)
```

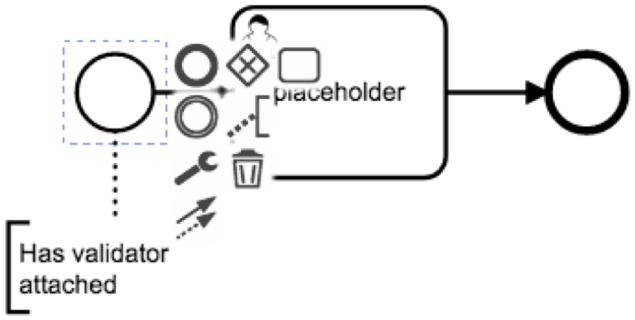
Public and Private keys locations are set in the ENV vars, but of course there are other JS functions in “rsaEncrypt.js” to set your own

API Form Validations Scripting

- Camunda BPM
- Variables that cannot be modified

POST JSON Payload ->

```
{
...
    "variables": {
        "age": {
            "value": 25,
            "type": "Integer"
        },
        "priority": {
            "value": 10,
            "type": "Integer"
        }
    },
...
}
```



Properties Panel

Form Key

Form Fields

server_validation1
validator2

Business Key

Form Field

ID

server_validation1

Type

string

Label

Default Value

Validation

Add Constraint

Name	Config
validator	io.digitalstate.camunda.JsFormValidation

Properties

Add Property

Id	Value
validator_file	form-validation.js

```
load('classpath:validationResult.js')
load('classpath:validate.min.js')
var JSONObject = Java.type('org.camunda.bpm.engine.impl.util.json.JSONObject')

var jsonSubmission = JSON.parse(new JSONObject(submissionValues).toString())

// Validate.js Constraints
function getConstraints() {
  var constraints = {
    age: {
      presence: true,
      numericality: {
        onlyInteger: true,
        greaterThan: 18,
        lessThanOrEqualTo: 125,
      }
    }
  };
  return constraints
}

// List of fields that are not allowed to be changed
function bannedFields(){
  // could also be loaded from another location (like a yaml or json file)
  return [
    "risk",
    "owner",
    "master_field",
```

Regular JS with Validate.js

```
19.   return constraints
20. }
21.
22. // List of fields that are not allowed to be changed
23. function bannedFields(){
24.   // could also be loaded from another location (like a yaml or json file)
25.   return [
26.     "risk",
27.     "owner",
28.     "master_field",
29.     "current_state",
30.     "flagged",
31.     "priority"
32.   ]
33. }
34.
35. // Check if the submission has any of the banned fields
36. function checkForBannedFields(submission, bannedFields) {
37.   // Loop through list of banned fields (nashorn loop)
38.   for each (var field in bannedFields){
39.     var hasBannedField = validate.contains(submission, field)
40.     // if a banned field was found then return true:
41.     if (hasBannedField == true) {
42.       return true
43.     }
44.   }
45.   // If no banned fields were found:
46.   return false
47. }
48.
```

```
45.   // If no banned fields were found:
46.   return false
47. }
48.
49. // if there is a banned field:
50. if (checkForBannedFields(jsonSubmission, bannedFields())){
51.   validationResult(false, {
52.                           "detail": "VALIDATE.JS",
53.                           "message": 'Submission contains a banned field: ' +
    JSON.stringify(bannedFields())
54.                         }
55.                   )
56.
57. // If no banned fields were found:
58. } else {
59.   // If no banned fields then continue:
60.   // Run Validations against Validate.js
61.   var validation = validate(jsonSubmission, getConstraints())
62.
63.   if (validation === undefined) {
64.     validationResult(true)
65.   } else {
66.     validationResult(false, {
67.                               "detail": "VALIDATE.JS",
68.                               "message": JSON.stringify(validation)
69.                             }
70.                     )
71.   }
72. }
73.
```

Validation Result helper

```
function validationResult(result, validationError) {
  if (result != true){
    result = false
  }

  if (result == true){
    return Java.asJSONCompatible({
      "result": true
    })
  } else {
    return Java.asJSONCompatible({
      "result": false,
      "validation_error": {
        "detail": validationError.detail,
        "message": validationError.message
      }
    })
  }
}

```

Unit Testing / Writing Tests as Script

- Spock Unit Testing (Groovy based)
- Eval Javascript
- Call specific functions from the Eval
- Mock the Java classes
- Use a more business friendly language

Image: https://tech.smartling.com/the-spock-epic-part-3-the-undiscovered-groovy-4b744815aa21

Spock Unit Testing of Nashorn JS


```
1.  @Shared ScriptEngine engine = new ScriptEngineManager().getEngineByName('nashorn');
2.
3.    //
4.    // gateway_decision.js
5.    //
6.    def 'Nashorn: gateway_decision.js'(int number_dataTable, boolean gatewayDecision) {
7.      setup:_ 'Setup Mocks and binding'

8.        def execution = Mock(DelegateExecution)
9.
10.       execution.getVariable('number') >> number_dataTable
11.
12.       engine.put('execution', execution)
13.

14.     when:_ 'Execute Script'
15.       // Gets the specific .js script as text/string.
16.       def source = this.class.getResource('bpmn/end-to-end/gateway_decision.js').text
17.


18.       def evalResult = engine.eval(source);
19.
20.     then:_ 'Gateway Decision matches expection'
21.
22.       assertThat 'The script retuned the exepected result',
23.                  evalResult == gatewayDecision
24.       println "Script Response: ${evalResult}"
25.       println "DataTable Expected Response: ${gatewayDecision}"
```

```
17.
18.       def evalResult = engine.eval(source);
19.
20.     then:_ 'Gateway Decision matches expection'
21.
22.       assertThat 'The script retuned the exepected result',
23.                  evalResult == gatewayDecision
24.       println "Script Response: ${evalResult}"
25.       println "DataTable Expected Response: ${gatewayDecision}"
26.
27.     where:
28.         number_dataTable || gatewayDecision
29.                       -1 || false
30.                        0 || false
31.                        1 || false
32.                        2 || false
33.                        3 || false
34.                        4 || false
35.                        5 || true
36.                        6 || true
37.                        7 || true
38.                        8 || true
39.                        9 || true
40.                       10 || true
41.                       11 || false
42.   }
43.
```

Incident Handlers

- Something goes wrong: a “Incident is created”
- “Every time in all processes, if something goes wrong, I want to Do XYZ”
- “notify by email when XYZ occurs/incident occurs”

- Multiple functions are possible:
  - Handle Incident
  - Resolve Incident
  - Delete Incident

Additional data about the incident is valuable during incident scripting: Use Bindings!

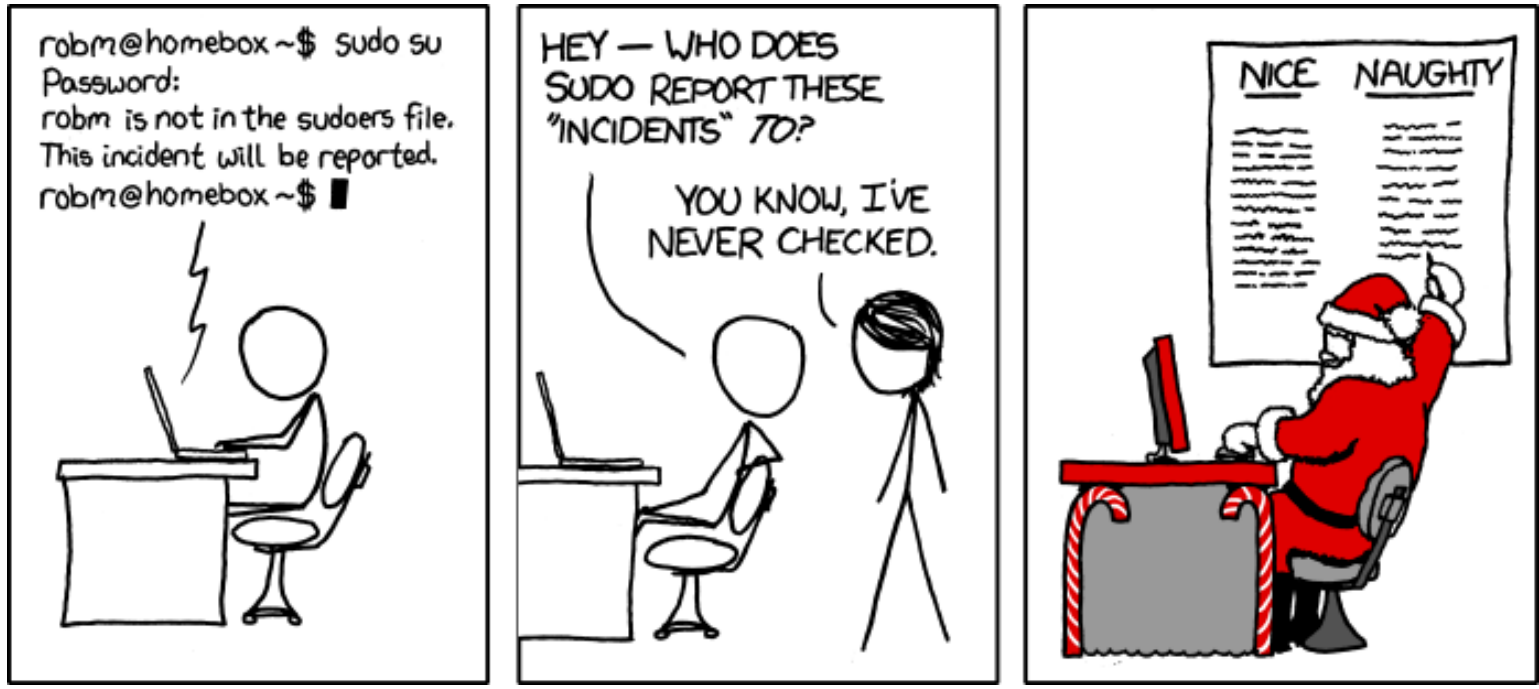
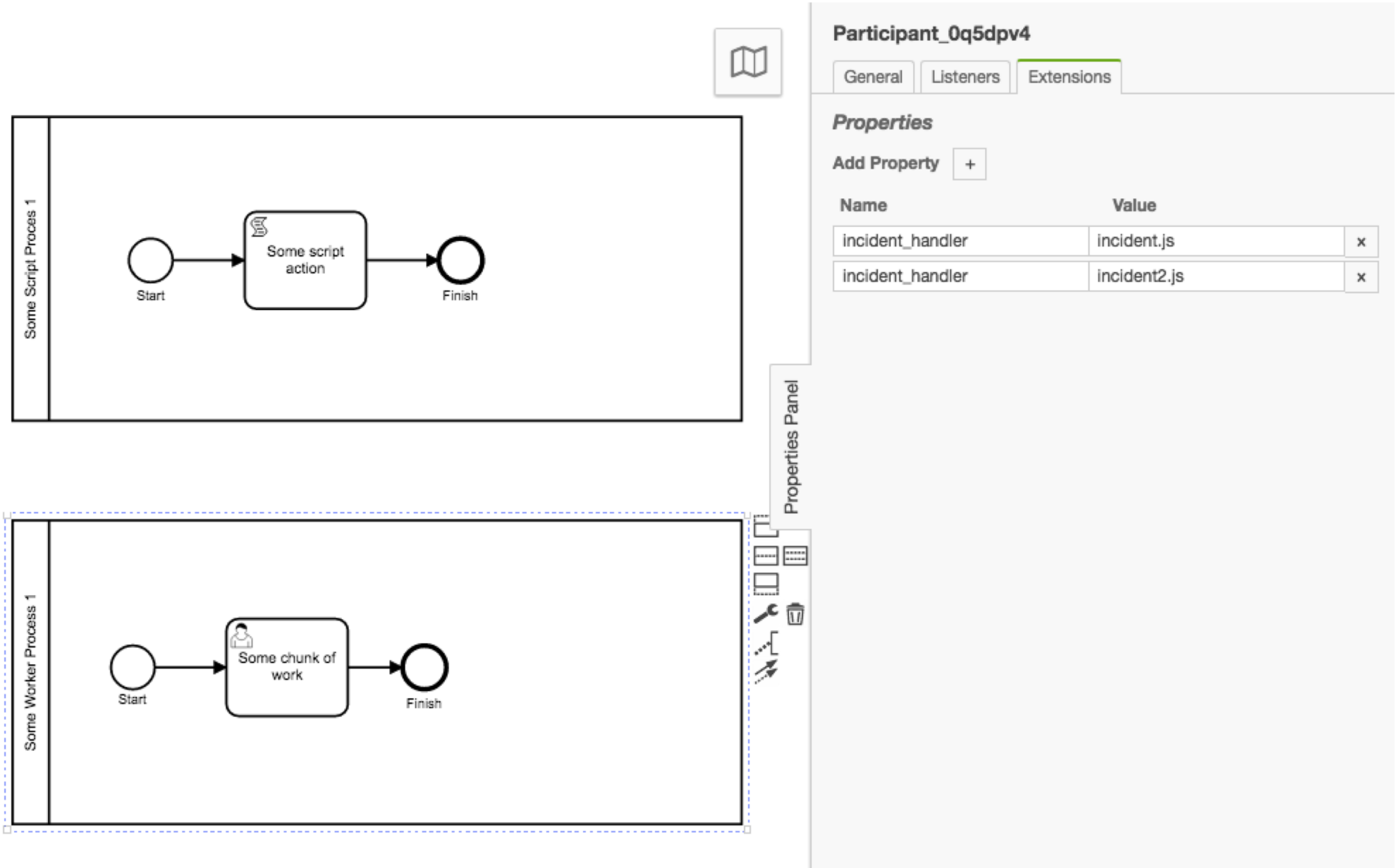


Image: https://xkcd.com/838/


Some Script Proces 1
Start
Some script action
Finish
Some Worker Process 1
Start
Some chunk of work
Finish
Properties Panel
Participant_0q5dpv4
General
Listeners
Extensions
Properties
Add Property
+
Name
Value
incident_handler
incident.js
incident_handler
incident2.js

Invoking a specific function


```
var system = java.lang.System

function handleIncident(){
  system.out.println("This is a handleIncident Function Message: ");
  system.out.println(incidentContext.getProcessDefinitionId())
  system.out.println(incidentMessage)

}

function resolveIncident(){
  system.out.println("This is executed when the incident was resolved");
}

function deleteIncident(){
  system.out.println("This is executed when the incident was deleted");
}


// incidentContext.getProcessDefinitionId()
// incidentContext.getActivityId()
// incidentContext.getExecutionId()
// incidentContext.getConfiguration()
// incidentContext.getTenantId()
// incidentContext.getJobDefinitionId()

// execution variable is available

// incidentMessage is also available
```

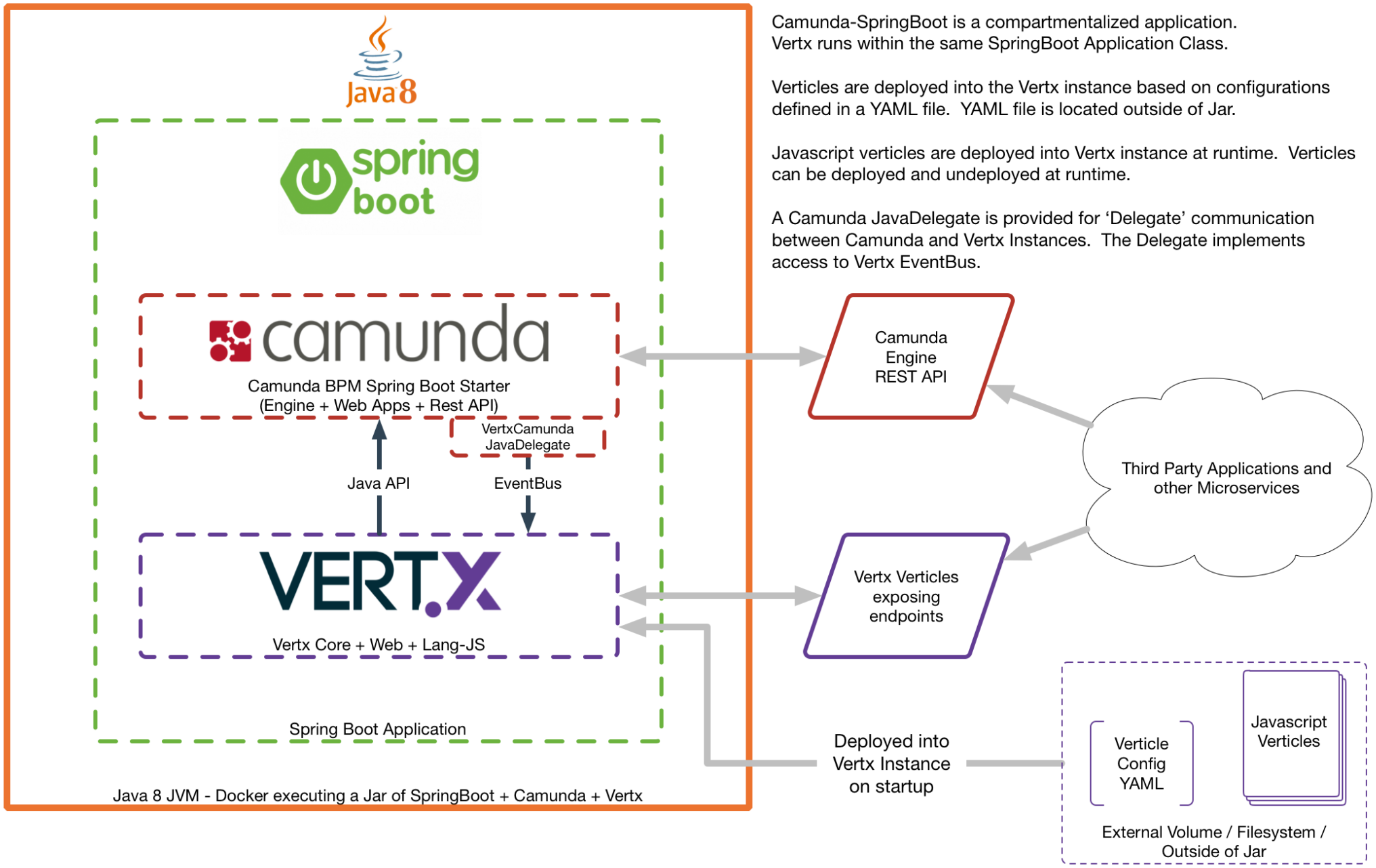
```
13.
14. function deleteIncident(){
15.   system.out.println("This is executed when the incident was deleted");
16. }
17.


18. // incidentContext.getProcessDefinitionId()
19. // incidentContext.getActivityId()
20. // incidentContext.getExecutionId()
21. // incidentContext.getConfiguration()
22. // incidentContext.getTenantId()
23. // incidentContext.getJobDefinitionId()
24.
25. // execution variable is available
26.
27. // incidentMessage is also available
28.

29. // example output
30. // camunda_1  | This is a message we wanted to see!!
31. // camunda_1  | myProcess-incident1:4:2c726ce0-4b60-11e8-8225-0242ac130002
32. // camunda_1  | ENGINE-09027 Exception while resolving duedate 'dog':
    Invalid format: "dog"
33.
```

Extend the Camunda API with Vertx: Camunda Plugin and Spring Boot

- With Tomcat or whichever container or with Springboot…
- Don’t want to write Jax-rs JARs and deploy.
- Endpoints specific to a Process instance (complete a task through email link)
- Technical staff can add new Vertx Verticles as they see fit.


Java 8
spring boot
camunda
Camunda BPM Spring Boot Starter
(Engine + Web Apps + Rest API)
VertxCamunda
JavaDelegate
Java API
EventBus
VERT.X
Vertx Core + Web + Lang-JS
Spring Boot Application
Java 8 JVM - Docker executing a Jar of SpringBoot + Camunda + Vertx
Camunda-SpringBoot is a compartmentalized application.
Vertx runs within the same SpringBoot Application Class.
Verticles are deployed into the Vertx instance based on configurations defined in a YAML file. YAML file is located outside of Jar.
Javascript verticles are deployed into Vertx instance at runtime. Verticles can be deployed and undeployed at runtime.
A Camunda JavaDelegate is provided for 'Delegate' communication between Camunda and Vertx Instances. The Delegate implements access to Vertx EventBus.
Camunda Engine REST API
Third Party Applications and other Microservices
Vertx Verticles exposing endpoints
Deployed into Vertx Instance on startup
Verticle Config YAML
Javascript Verticles
External Volume / Filesystem / Outside of Jar

BPMN Documentation Generator: Camunda BPMN Model Library, Freemarker and Vertx

```
function instanceClass(type){
    var bpmnInstanceClass = Java.type('org.camunda.bpm.model.bpmn.instance.' + type).class
    return bpmnInstanceClass
}
```

Freemarker Template

```
1. <#list tasks_>
2. Elements:
3. <#items as task>
4. Task Type:
5. ${task.getElementType().getTypeName()}
6.
   Task Name:
7. ${task.getName()}
8.
   Custom Attributes:
9. <#list task.getExtensionElements().getElementsQuery().filterByType(extensionElements_).singleRes
10. <#items as extensionElement>
11. Attribute:
12. Key: ${extensionElement.getCamundaName()}
```

```
lame()}


s().getElementsQuery().filterByType(extensionElements_).singleResult().getCamundaProperties()>


ndaName()}
undaValue()}
```

```
<#list tasks_>
Elements:
<#items as task>
Task Type:
${task.getElementType().getTypeName()}

Task Name:
${task.getName()}

Custom Attributes:
<#list task.getExtensionElements().getElementsQuery().filterByType(extensionElements_).singleRes
<#items as extensionElement>
Attribute:
Key: ${extensionElement.getCamundaName()}
Value: ${extensionElement.getCamundaValue()}

</#items>
</#list>
--------------------

</#items>
</#list>
```

Task_0b7n9ji

General | Forms | Listeners | Input/Output | Extensions

Properties

Add Property +

Name	Value	
order	some order value	×
title	some title value	×
failure	some fail	×
resolution	some resolution	×

Properties Panel

Freemarker Template output

```
Elements:
  Task Type:
  userTask

  Task Name:
  Step 1

  Custom Attributes:
      Attribute:
      Key: order
      Value: some order value

      Attribute:
      Key: title
      Value: some title value

      Attribute:
      Key: failure
      Value: some fail

      Attribute:
```

```
        Key: order
        Value: some order value

        Attribute:
        Key: title
        Value: some title value

        Attribute:
        Key: failure
        Value: some fail

        Attribute:
        Key: resolution
        Value: some resolution

--------------------

  Task Type:
  userTask
```

Enterprises, governments and those difficult app environments

Reaction they give you when starting the conversation about JS and Java

Large Niche for Nashorn/JVM JS execution

Dynamic, runtime extensions of your Java applications.

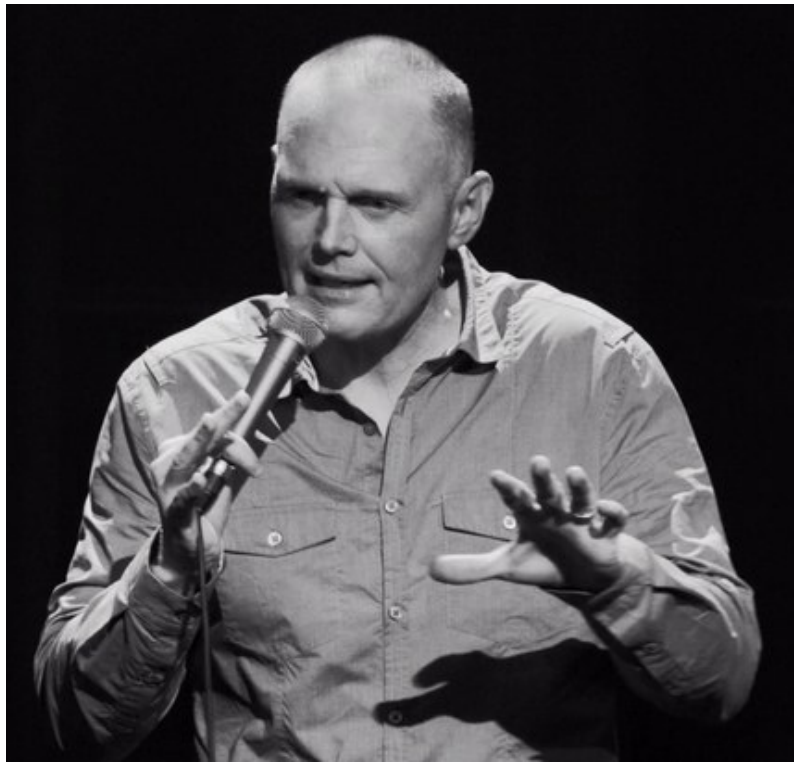
Add new functionality to applications that the technical business users want to modify. Netflix example.

Alternative to: “Why not just rebuild/rewrite in node?...”

Vertx: Pick the language best suited without the overhead

Enterprises, governments and those difficult app environments

Reaction they give you when starting the conversation about JS and Java



You: Stay in the Pocket.
You got this.
Stay with me..

enterprise, Gov, or whatever “difficult env: already using Java?

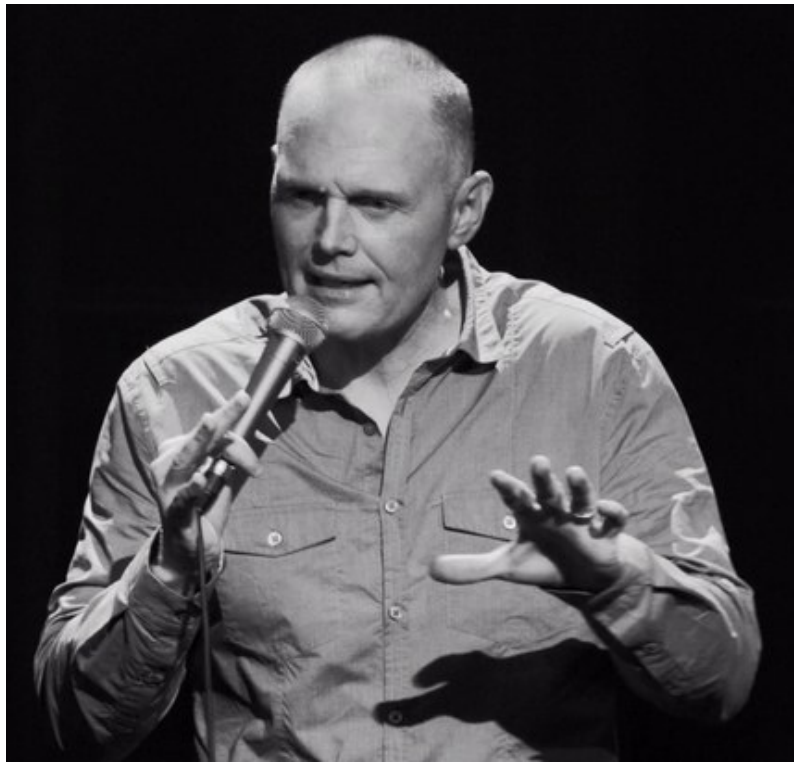
There is still options to use Javascript within the approved JVM.

Enable technical business staff to write the scripting layer.

It’s Java.. But it’s also JS.

Enterprises, governments and those difficult app environments

Reaction they give you when starting the conversation about JS and Java



You: Stay in the Pocket.
You got this.
Stay with me..

You: When you are **quickly** producing functionality in that “Java App” that no one wanted to touch, recompile, deploy, etc

Thank you

Underrated

Nashorn Javascript

Presented by Stephen Russett
@stephenrussett
Github: StephenOTT